UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

PROYECTO FIN DE CARRERA

"Implementación de Bloques de Simulink para Control Adaptativo de un Motor de Corriente Continua"

Ingeniería Industrial Especialidad en Electrónica y Automática

Autor	**|**	David Rodríguez Rosa
Tutores	**|**	Ramón Ignacio Barber Castaño
	|	Santiago Garrido Bullón

Leganés, 3 de octubre de 2012

Título: *Implementación de Bloques de Simulink para Control Adaptativo de un Motor de Corriente Continua.*
Autor: *David Rodríguez Rosa.*
Tutores: *Ramón Ignacio Barber Castaño y Santiago Garrido Bullón*

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de _____.

VOCAL

SECRETARIO PRESIDENTE

A mi abuelo que me enseñó a seguir luchando cuando la razón dicta que es hora de rendirse y a mi familia que siempre ha estado junto a mí.

A Ángela y a Adriana porque cuando miro junto a ellas el mundo, todo parece más sencillo.

A Ismael y a Luis por su incansable empeño en hacer realidad mis sueños.

A los que compartieron mis mismas ilusiones, las noches de desvelo y las tardes de risas que desdibujaron nuestras preocupaciones.

A Ramón por darme la oportunidad de poder trabajar en este proyecto del que tanto he aprendido y a Santiago por toda su ayuda.

A todas aquellas vidas que se cruzaron en la mía y que, para bien o para mal, han hecho que me encuentre en este final, agradeciendo hoy, que pueda tener a alguien a quien agradecérselo.

… y a mis padres, por no tener ni si quiera palabras con las que poder expresárselo.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1. Introducción

Desde el comienzo de la humanidad el hombre ha buscado diversas formas de mejorar su vida y hacerla más sencilla intentando encontrar sistemas que realizasen de forma automática tareas pesadas o rutinarias *[1]*. Ya en la antigua Grecia se comienza a vislumbrar los primeros sistemas de control de cierta relevancia, con la aparición de ingenios mecánicos capaces de funcionar por sí mismos. Claro es el ejemplo del reloj de agua de Ktesibios, en el siglo III antes de Cristo, o el sistema de regulación de una lámpara de aceite de Philon de Bizancio, el cual suministraba más aceite al depósito principal de la lámpara, desde otro depósito secundario, según este se iba consumiendo.

Los primeros sistemas realimentados surgen en el siglo I antes de Cristo, entre las páginas de un libro escrito por Herón de Alejandría para su Enciclopedia Técnica llamado "Pneumática". Uno de estos ingenios mecánicos, el cual se describía en su libro, era un dispensador de vino que funcionaba mediante la teoría de vasos comunicantes.

Aunque existió un gran desarrollo y mejoras técnicas, no hubo demasiados avances en sistemas realimentados hasta el siglo XVII donde se presentaron distintos sistemas de regulación de temperatura, salvo la excepción importante del molino de H.U. Lansperg hacia el año 1200, que no molía a velocidad constante si no que dependía de otros factores como la fuerza del viento o la dureza del propio grano.

Es con la revolución industrial y el uso de la máquina de vapor cuando se pudo ver el primer regulador con un uso real que incorporaba sensor y actuador en el mismo dispositivo, gracias a la aparición del regulador de Watt. Este regulador mecánico controlaba la cantidad de vapor que salía de la caldera a la turbina de la máquina de vapor suministrando una acción proporcional y debía su gran estabilidad al elevado índice de rozamiento. Este tipo de reguladores fueron denominados posteriormente moderadores (Reguladores de acción proporcional) por Maxell, y no controladores.

Aún con la existencia de estos ejemplos no se puede decir que aún hubiese una Ingeniería o Teoría de Control como tal, pues entre otras cosas faltaban las herramientas matemáticas para ello. Es por la misma época en la que Watt perfeccionaba su regulador, cuando Laplace y Fourier desarrollaban métodos de Transformación matemática, que unidos a formulación de la teoría de la variable compleja de Cauchy, asentarían las bases matemáticas necesarias para la Ingeniería de Control.

Con la evolución de la máquina de vapor también evolucionó el regulador de Watt y las técnicas de diseño, provocando una disminución del rozamiento del regulador lo que conllevó comportamientos indeseados en el conjunto del sistema. El problema fue resuelto por Maxwell y Vischnegradsky, que publicaron sus resultados. Además Vischnegradsky demostró

que no era posible controlar un sistema de tercer orden mediante un regulador proporcional. Este hecho, junto a la publicación de Maxell en 1868 de "On Governors", que establece la diferencia entre reguladores proporcionales e integrales, marca el origen de la Teoría de Control.

A partir aquí, y con unos sólidos cimientos sobre los que seguir trabajando, grandes pensadores como Liapunov, Routh, Nyquist, etc. han ido haciendo su aporte hasta conseguir la Ingeniería de Control que se conoce hoy en día. Sin embargo los avances tecnológicos y las nuevas necesidades de control, hacen que se requiera un grado superior de flexibilidad ante la variabilidad de los sistemas reales, jugando en este punto un papel muy importante los controladores adaptativos, marcando el presente y el futuro de la ingeniería de control.

1.1. Motivación

El presente proyecto pretende obtener controladores adaptativos capaces de interactuar con un sistema físico y regularlo en tiempo real, para lo que se hará uso de una maqueta con un motor de corriente continua para realizar las pruebas y obtener los resultados experimentales, aunque después será posible portarlo a cualquier sistema realizando una serie de cambios mínimos. Para poder alcanzar este fin se utilizará MATLAB y el entorno de simulación Simulink, donde se crearán los modelos necesarios a través de su lenguaje de bloques.

En general, el control mediante uso de reguladores convencionales es la práctica más común y extendida quizás por su mayor grado de implantación debido a su madurez frente a los controles adaptativos, cuyo análisis y aplicación es más reciente. La implementación de controladores convencionales y su funcionamiento, tras la realización de un estudio previo del sistema, suelen dar muy buenos resultados, sin embargo están sujetos a ciertas limitaciones que se pueden evitar gracias al control adaptativo *[25]*. Una de estas grandes limitaciones es el hecho de que la identificación del sistema real y los cálculos de los parámetros del controlador se realizan fuera de línea, por lo que no linealidades en los valores del sistema provocan inestabilidades en el conjunto o un funcionamiento del sistema completo que no fue el concebido en su diseño. De esta forma el control adaptativo tiene la ventaja de trabajar en línea, por lo que los parámetros de la planta los identifica en cada instante y con ellos va recalculando los del controlador. Aun así hay que decir que el control adaptativo requiere una mayor complejidad de diseño, por lo que se debe plantear si realmente es necesaria su aplicación, estudiándose con detenimiento cada caso concreto y evaluándose los pros y contras de su utilización.

MATLAB y concretamente el entorno de Simulación de Simulink, proporciona una potente herramienta con un entorno gráfico interactivo para poder realizar la tarea de una manera sencilla. Además la incorporación de los módulos de Real-Time Windows Target permite control en tiempo real, ideal desde un punto de vista de control y simulación HIL (Hardware-in-the-loop). Sin embargo el potencial de esta herramienta no se queda ahí, existiendo la posibilidad de exportar estos modelos a otros lenguajes de programación o de crear

ejecutables que trabajen bajo el sistema operativo de Windows, por lo que no es únicamente un trabajo desde un punto meramente educativo o que está sometido a un reducido número de pruebas en un laboratorio, sino que luego puede ser aplicable en los entornos reales donde sea necesario.

Aunque posiblemente la implementación de controladores adaptativos era posible antes de la realización de este proyecto, haciéndose uso de las mismas herramientas, surgía un problema difícilmente salvable, la gestión de los estados discretos en tiempo real cuando se trabajaba en *External Mode*, ya que aun existiendo formas de hacerlo, era una tarea compleja. A partir de la versión 2012a de MATLAB, aparecen una serie de facilidades que hacen que este trabajo, en un principio tan complicado, se vuelva más factible de realizar.

Se puede decir que se está en el momento idóneo para el comienzo de la implementación y el estudio de controladores adaptativos haciendo uso para ello de la herramienta Simulink de MATLAB, trabajando con estos controladores en tiempo real y en modo externo. Así, en resumen, el objetivo principal que pretende alcanzase con este proyecto no es otro que dar ese primer paso, demostrando que realmente esto puede llevarse a cabo de una manera práctica y viable, obteniéndose además con ello unos resultados más que aceptables.

1.2. Objetivos

El proyecto en sí no pretende tanto la implementación de controladores adaptativos concretos, sino más bien demostrar la posibilidad de que a día de hoy es posible implementarlos con MATLAB de una forma relativamente sencilla, haciendo uso para ello de un pequeño número de controladores a los que se los hará trabajar de manera adaptativa a forma de ejemplos ilustrativos.

De esta manera como objetivo principal del proyecto se ha establecido:

- Implementación de controladores adaptativos en Simulink trabajando en sistemas físicos.

Este objetivo se desglosa en:

- Estudio de las nuevas posibilidades, tras la inclusión de bloques de funciones y S-funciones en las librerías de Simulink, para Real-Time Windows target.
- Implementación de reguladores adaptativos.
- Estudio sobre un sistema real.

Esto es posible gracias a los bloques *"S-Function Builder"* que son capaces de generar de forma automática el archivo .tlc. Estos bloques y su posibilidad de generar código de forma automática es una incorporación que MATLAB realizó en las librerías de Simulink a partir de su versión 2012a de la suite, solventando lo que en un principio era un gran problema si se

necesitaba gestionar estados discretos, sobre todo si estos estados dependían de valores pasados.

Aun teniéndose esta herramienta que, aunque con ciertas carencias, simplifica en gran medida la tarea abordada, todavía existían una serie de barreras que debían ser solventadas antes de comenzar a ver que esta implementación era posible. Para estudiar la viabilidad final se siguieron una serie de pasos iniciales:

> ***Paso 0 –*** Pruebas con sistemas continuos y discretos en *Normal Mode* y *External Mode*, comprobando que funcionan correctamente, y viéndose que configuración es la requerida para conseguir este funcionamiento.
>
> ***Paso 1 –*** Pruebas con los distintos bloques de funciones de Simulink, comprobando ventajas y desventajas de cada uno de aquellos que sean compatibles con el *External Mode*.
>
> ***Paso 2 –*** Creación de esquemas sencillos de control que permitan comprobar si es posible la modificación de parámetros en tiempo de ejecución haciéndose uso de los distintos bloques función que sean capaces de trabajar de forma correcta en *External Mode*.

Una vez llegados a este punto se ha podido comprobar que la implementación de los controladores adaptativos con Simulink en tiempo real y en *External Mode* es viable, por lo que los siguientes pasos ya están encaminados a implementarlos. Para ello se requerirá:

> ***Estudios teóricos iniciales –*** Conocimiento previo teórico sobre métodos de identificación necesarios para su implementación práctica posterior, comparándolos y viendo los más apropiados para el control del sistema real. Elección de los reguladores a implementar, estudio teórico de estos y obtención de sus ecuaciones.
>
> ***Identificación fuera de línea –*** Identificación fuera de línea del sistema real (en este caso se ha usada una maqueta con un motor de corriente continua) para poder hallar posteriormente los valores de los controladores convencionales cuyos parámetros son fijos y toman el control en los instantes iniciales.
>
> ***Identificación en línea –*** Implementación del método de identificación seleccionado que se encarga de la estimación de los valores de la planta.
>
> ***Implementación de los controladores convencionales iniciales –*** Implementación de diversos controladores convencionales (PID, Dead-beat) para los valores obtenidos en la identificación del sistema fuera de línea. Estos controladores sirven para tener una referencia de los valores de los controladores convencionales que deberán actuar antes que los adaptativos a la espera de que el sistema completo arranque y se estimen unos valores correctos de la planta.

Implementación de controladores adaptativos **–** Esta implementación consistirá en la unión de todos los recursos obtenidos en los distintos pasos seguidos para llegar hasta aquí, y se dividirá, como se muestra en la figura ***Figura 1.1***, en cuatro pasos básicos. El primero solo se realiza en un periodo inicial de tiempo y consiste en la actuación del control convencional mientras que se obtiene una primera estimación del sistema. A partir de ahí se comienza un proceso iterativo de identificación del sistema, cálculo de los parámetros del regulador y actuación del control adaptativo con los parámetros calculados.

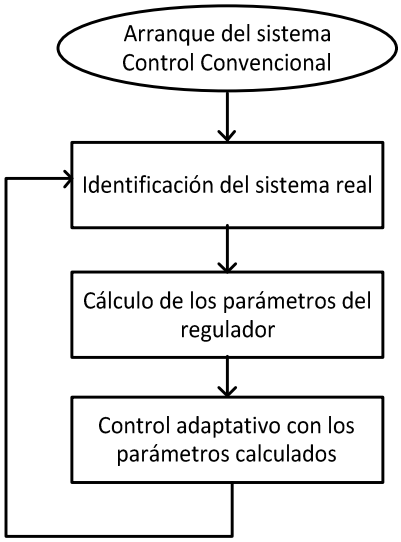

Figura 1.1. Esquema básico de funcionamiento del control adaptativo

Así el resultado obtenido es un control adaptativo que es capaz de calcular los valores del controlador en cada instante tiempo en función de las respuestas del sistema que está controlando, lo que produce un ajuste preciso, evitando comportamientos inestables o no deseados en caso de cambios no lineales en los parámetros de la planta.

1.3. Partes del documento

El documento consta de los capítulos que se describen a continuación:

Capítulo 1. Introducción

El presente capítulo pretende dar una primera visión del proyecto y de lo que se desea alcanzar con él. Para ello se hace una breve introducción que muestra el interés y necesidad de los sistemas de control a través de la historia, produciéndose con ellos una mejora en la calidad de vida. Además se describe la motivación que llevó a realizar este proyecto y los objetivos que se han marcado, todo esto junto a una breve introducción al contenido de este documento.

Capítulo 2. Identificación y Control Adaptativo

En este capítulo se verá en qué consiste la identificación de sistemas, abordándose algunos métodos de identificación para llevar a cabo el proyecto, como son el método de mínimos cuadrados, el método de mínimos cuadrados recursivo y el método de mínimos cuadrados recursivo con factor de olvido. Además se hará una introducción al concepto de Control Adaptativo, viendo en que consiste, su esquema básico de actuación y la parte teórica necesaria de algunos controles como son el PID o el Dead-Beat para poder realizar posteriormente su implementación con Simulink.

Capítulo 3. Control con Simulink en External Mode

En este punto se describirá el software elegido para llevar a cabo la implementación de los controles adaptativos, en este caso Simulink de MATLAB, haciéndose un repaso por los conceptos necesarios para comprender el funcionamiento de este y de sus componentes. También se ahondará en las distintas formas que el usuario dispone para poder crear funciones definidas por el mismo (Fcn, MATLAB Function, S-Function y S-Function Builder), y las ventajas que presentan unas respecto a las otras. Todo esto sin perder de vista que la implementación final debe ser acabada en *External Mode* para conseguirse una interactuación en tiempo real con el sistema físico.

Capítulo 4. Implementación

Se explicarán los pasos necesarios para conseguir la implementación de un controlador adaptativo, como son la configuración del entorno de trabajo, las identificaciones del sistema tanto fuera de línea, para poder obtener posteriormente los parámetros del controlador convencional inicial, como la identificación en línea, para ser usada en el control adaptativo, y la creación del control adaptativo usando para ello como casos prácticos un regulador PID y un controlador Dead-Beat, desglosando ambos casos en el cálculo del regulador convencional y la implementación propiamente dicha de los controladores adaptativos.

Capítulo 5. Resultados experimentales

Se mostrará la respuesta de un PID clásico a cambios en los parámetros de la planta y los resultados obtenidos en la identificación fuera de línea para cada uno de los reguladores implementados. Se comprobará si el comportamiento de estos controladores adaptativos era el esperado y si para los distintos valores de la constante proporcional k del motor la dinámica de control permanece constante. Además se verá si son capaces de adaptarse a cambios bruscos de k adecuando la dinámica del sistema. También se realizará una comparativa entre la respuesta de un PID convencional y un PID adaptativo.

Capítulo 6. Conclusiones y trabajos futuros

Se expondrán las conclusiones finales obtenidas de la realización del proyecto y los trabajos futuros con el fin de mejorar ciertos puntos estudiados para los controles adaptativos que se verán, y otras posibles vías de control adaptativo.

Anexos

Contendrá los modelos implementados y los fragmentos de código que por su extensión o complejidad podrían hacer que al ser introducido con el resto del documento se diluyese, en parte, el fin que se pretende alcanzar con él.

Capítulo 2. Identificación y Control Adaptativo.

La identificación y control de sistemas permite la posibilidad de identificar y modelar matemáticamente sistemas con el fin de obtener el comportamiento deseado en ellos, de la forma más exacta posible, gracias a un controlador que será el encargado de modificar la dinámica del conjunto.

A la hora de obtenerse este modelo matemático *[2]*, hay dos opciones:

- ***Fuera de línea –*** Se obtiene un modelo matemático como producto de técnicas de identificación que dan como resultado el diseño de un regulador con parámetros fijos, ya que estos parámetros no están sujetos a restricciones temporales. Este proceso debe ser repetido para conseguir un ajuste lo más exacto posible de estos parámetros.
- ***En línea –*** Se obtiene un modelo matemático resultado de técnicas de identificación en cada periodo de muestreo, por lo que el resultado es el diseño de un regulador, con una serie de parámetros que pueden variar en el tiempo con cada nuevo modelo matemático. Estos modelos deben ajustarse a una estructura definida previamente. Así, con esta opción, se tiene la posibilidad de implementar Controles Adaptativos calculando los parámetros de estos en cada instante de muestreo.

2.1. Identificación de procesos

La identificación de procesos *[4]* consiste en su representación matemática mediante un conjunto de estudios, teorías y algoritmos con un cierto grado de exactitud. Por norma general, cuando se habla de identificación se hace referencia a sistemas que están siendo modelados a través de la experimentación sobre el propio sistema, al no existir unas ecuaciones físicas deducibles o estas sean difíciles de obtener, mientras que en el modelado de sistemas se llega a un modelo teórico a través de la formulación de ecuaciones que permiten describir la forma de este. Claramente se puede ver que los modelos que se obtienen por identificación son modelos de caja negra, de los que no se sabe cuál es su forma real ni el valor de sus parámetros, o de caja gris, en los que se sabe cuál podría ser su forma pero no el valor que pueden tomar dichos parámetros, pero que a través de su respuesta ante una entrada se los puede identificar, obteniéndose una expresión matemática que se ajuste a este comportamiento, no teniendo porque ser esta expresión la real que les define y que representa de forma bastante fiel al sistema real que se está identificando.

Para poder llevarse a cabo la identificación de procesos se debe tener acceso a una serie de datos, como son las entradas y salidas del sistema a identificar, se debe disponer de varios modelos matemáticos a los que se puedan ajustar estos datos y por último determinar cuál de los modelos se adapta mejor, en función de los datos, haciéndose uso de algún tipo de criterio. Una vez llegado a este punto se debe comprobar que el modelo alcanza las expectativas iniciales. A estas comprobaciones se le denominan validación del modelo.

Se utilizan en la identificación de procesos del proyecto métodos paramétricos, en los que el modelo matemático que describe el funcionamiento del sistema tiene un conjunto finito de coeficientes. Los métodos más comunes por lo general se basan en la minimización del error de predicción.

Los métodos paramétricos, como los mínimos cuadrados que se verán más adelante, se basan en la predicción del error (la diferencia entre la salida del proceso y la predicción hecha por el modelo). Así es imprescindible de disponer de un modelo con el que describir el comportamiento del sistema. En el caso del presente proyecto se hará uso de un modelo ARX por sencillez.

2.1.1. Modelo ARX (auto-regressive and exogenous variable)

Este modelo *[4]* es el más simple con el que se puede describir el comportamiento dinámico de un conjunto discreto de mediciones de entrada y salida tomadas durante un cierto número de intervalos de tiempo T, representandose con la siguiente expresión:

$$y(k) + a_1 y(k-1) + \cdots + a_{n_a} y(k-n) = b_1 u(k-1) + \cdots + b_{n_b} u(k-n) + e(k) \quad \textbf{(2.1)}$$

donde $e(k)$ *[2]* es un ruido blanco que entra como error directo. Debido a que es un error no medible, su cálculo, a partir del modelo, puede servir como una estimación de este. En muchos casos asumir o no este error, aleja o acerca a la identificación del proceso de la realidad. Añadiéndole se asumen factores tales como las no linealidades o la imprecisión en las mediciones que influyen sobre la salida. Así este error engloba este grupo de comportamientos, aceptando que la parte determinista del modelo no es exacta.

Con esto, el vector de parámetros ajustables del modelo θ es:

$$\theta = [a_{1,} \dots, a_{n_a}, \; b_1, \dots, b_{n_b}]^T \quad \textbf{(2.2)}$$

Si se definen además los polinomios:

$$A(z) = 1 + a_1 z^{-1} + \cdots + a_{n_a} z^{-n_a} \quad \textbf{(2.3)}$$

$$B(z) = b_1 z^{-1} + \cdots + a_{n_b} z^{-n_b} \quad \textbf{(2.4)}$$

se obtiene que:

$$G(z) = \frac{B(z)}{A(z)} \tag{2.5}$$

$$H(z) = \frac{1}{A(z)} \tag{2.6}$$

donde el modelo que representa se puede ver en la ***Figura 2.1.***

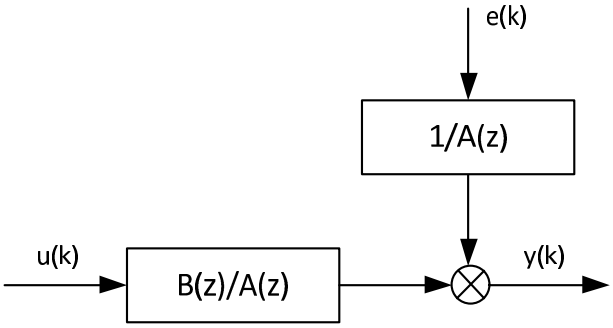

Figura 2.1. Estructura del modelo ARX

En el caso concreto de que $n_a = 0$ la salida es modelada como un sistema con respuesta impulsional finita.

2.1.2. Método de mínimos cuadrados

El pilar fundamental de este método *[4]* se centra en el concepto de regresión lineal, y permite la identificación en tiempo real de modelos tanto lineales como no lineales, siempre y cuando sus parámetros si lo sean.

Para ello se considera como modelo paramétrico, el modelo ARX (auto-regressive and exogenous variable) explicado anteriormente, pero sin tener en cuenta el ruido blanco que entra como error directo. Su expresión escrita en ecuaciones en diferencias queda:

$$y(k) + a_{1y}y(k-1) + \cdots + a_n y(k-n) = b_1 u(k-1) + \cdots + b_n u(k-n) \tag{2.7}$$

Con lo que se dispone de un modelo determinista cuya función de transferencia corresponde a la siguiente expresión:

$$G(z^{-1}) = \frac{b_1 z^{-1} + \cdots + b_n z^{-n}}{1 + a_1 z^{-1} + \cdots + a_n z^{-n}} \tag{2.8}$$

Se puede reescribir la ecuación ***(2.7)*** como:

$$y(k) = \varphi^T \theta \qquad \textbf{(2.9)}$$

dónde:

- $y(k)$ es la magnitud medible.
- φ es el vector de regresor:

$$\varphi(k) = [-y(k-1) \dots - y(k-n)\, u(k-1) \dots u(k-n)] \qquad \textbf{(2.10)}$$

- θ es el vector de parámetros:

$$\theta = [a_1 \dots\ a_n\ b_1 \dots\ b_n]^T \qquad \textbf{(2.11)}$$

De esta forma para el valor del vector θ en un cierto instante k se produce un error de predicción:

$$e(k,\theta) = y(k) - \hat{y}(k) = y(k) - \varphi(k)\theta \qquad \textbf{(2.12)}$$

Se puede observar por la expresión ***(2.9)*** que el número de incógnitas que se tiene es de $2n$ *[5]*, por lo que a partir de este número de medidas se puede determinar el valor real del vector θ al disponerse de las $2n$ ecuaciones necesarias. Sin embargo, debido al hecho de que el modelo que se intenta estimar no suele coincidir con el sistema real, el método de mínimos cuadrados parte de N pares $[y(k); \varphi(k)]$, un valor muy superior a $2n$, ya que no se puede encontrar de forma general un vector de parámetros que consiga hacer cero el error, en cambio sí que se puede hallar un vector que minimice el error, pues se trata de sistemas de ecuaciones incompatibles. En el caso, poco probable, de que coincidan proceso y modelo, se tiene un sistema sobredeterminado compatible, lo que da como resultado un error de predicción cero a partir de la muestra $2n$.

Las medidas obtenidas desde $k = n$, hasta $k = N$ se agrupan de la siguiente manera:

$$E(N,\theta) = Y(N) - \Phi(N)\theta \qquad \textbf{(2.13)}$$

donde los vectores $E(N,\theta)$ e $Y(N)$ tienen la siguiente composición:

$$E(N,\theta) = [e(n,\theta) \dots e(N,\theta)]^T \qquad \textbf{(2.14)}$$

$$Y(N) = [y(n) \dots y(N)]^T \qquad \textbf{(2.15)}$$

y estando compuesta la matriz $\Phi(N)$ por los regresores correspondientes:

$$\Phi(N) = \begin{bmatrix} \varphi(n) \\ \vdots \\ \varphi(N) \end{bmatrix} \tag{2.16}$$

De esta manera se define el índice de bondad de ajuste como:

$$J(\theta) = \|E(N,\theta)\|^2 = \sum_{k=n}^{N} e^2(k,\theta) \tag{2.17}$$

que se puede reescribir de la forma:

$$J(\theta) = [Y(N) - \Phi(N)\cdot\theta]^T \cdot [Y(N) - \Phi(N)\cdot\theta] \tag{2.18}$$

donde el mínimo valor de $J(\theta)$ se da en el valor del vector de parámetros que cumpla:

$$\frac{dJ(\theta)}{d\theta} = 0 \Longrightarrow 2\cdot[\Phi(N)\cdot\theta - Y(N)]^T \cdot \Phi(N) = 0 \tag{2.19}$$

obteniéndose que el valor del vector de parámetros, que hace mínimo este índice y por lo tanto pertenece al modelo identificado. es:

$$\hat{\theta} = [\Phi^T(N)\cdot\Phi(N)]^{-1}\cdot\Phi^T(N)Y(N) \tag{2.20}$$

2.1.3. Mínimos cuadrados recursivo

El método de Mínimos Cuadrados Recursivo permite estimar en tiempo real los parámetros del sistema, y puesto que deriva del método de mínimos cuadrados, está sujeto a las mismas restricciones.

La estimación obtenida en el instante k viene dada por la expresión:

$$\begin{aligned} \hat{\theta}(k) &= [\Phi^T(k)\cdot\Phi(k)]^{-1}\cdot\Phi^T(k)\cdot Y(k) = P(k)\cdot\Phi^T(k)\cdot Y(k) \\ &= P(k)\cdot[\Phi^T(k-1)\cdot Y(k-1) + \varphi^T(k)\cdot y(k)] \end{aligned} \tag{2.21}$$

donde $P(k)$ es la matriz de covarianza, de la que se puede comprobar que:

$$P^{-1}(k-1) = P^{-1}(k) - \varphi^T(k)\cdot\varphi(k) \tag{2.22}$$

Por otra parte se puede obtener:

$$\Phi^T(k-1)\cdot Y(k-1) = P^{-1}(k-1)\cdot\hat{\theta}(k-1) \\ = P^{-1}(k)\cdot\hat{\theta}(k-1) - \varphi^T(k)\cdot\varphi(k)\cdot\hat{\theta}(k-1) \quad (2.23)$$

Con lo que si se sustituye en la expresión ***(2.21)*** se obtiene:

$$\hat{\theta}(k) = \hat{\theta}(k-1) - P(k)\varphi^T(k)\cdot\varphi(k)\cdot\hat{\theta}(k-1) + P(k)\cdot\varphi^T(k)\cdot y(k) \\ = \hat{\theta}(k-1) - P(k)\cdot\varphi^T(k)\cdot\left[y(k) - \varphi(k)\cdot\hat{\theta}(k-1)\right] \quad (2.24)$$

De esta forma se ha expresado el vector de parámetros de una forma recursiva. Se pueden reescribir las ecuaciones *[3]* para que queden de la siguiente manera:

$$P(k) = \left[P(k-1) - \frac{P(k-1)\cdot\varphi(k)\cdot\varphi^T(k)\cdot P(k-1)}{1+\varphi^T(k)\cdot P(k-1)\cdot\varphi(k)}\right] \quad (2.25)$$

$$K(k) = \frac{P(k-1)}{1+\varphi^T(k)\cdot P(k-1)\cdot\varphi(k)} \quad (2.26)$$

$$e(k) = y(k) - \varphi^T(k)\cdot\hat{\theta}(k-1) \quad (2.27)$$

$$\hat{\theta}(k) = \hat{\theta}(k-1) + K(k)\cdot\varphi(k)\cdot e(k) \quad (2.28)$$

Obtenidas estas expresiones, los pasos que debe seguir el algoritmo de forma recursiva *[5]* son los siguientes:

- Inicialización de la matriz de covarianza P y el vector de parámetros θ.
- Calculo recursivo. Para cada instante k:
  - Obtener valores $y(k)$ y $u(k)$ de la planta (entrada y salida de esta).
  - Formar el vector de regresión $\varphi(k)$:

$$\varphi(k) = [-y(k-1)\ldots - y(k-n)\,u(k-1)\ldots u(k-n)] \quad (2.29)$$

  - Calcular la matriz de covarianza $P(k)$ mediante:

$$P(k) = \left[P(k-1) - \frac{P(k-1)\cdot\varphi(k)\cdot\varphi^T(k)\cdot P(k-1)}{1+\varphi^T(k)\cdot P(k-1)\cdot\varphi(k)}\right] \quad (2.30)$$

  - Calcular $K(k)$ según la expresión:

$$K(k) = \frac{P(k-1)}{1+\varphi^T(k)\cdot P(k-1)\cdot\varphi(k)} \quad (2.31)$$

  - Estimar el error cometido en el instante k:

$$e(k) = y(k) - \varphi^T(k)\cdot\hat{\theta}(k-1) \quad (2.32)$$

- Calcular $\hat{\theta}(k)$:

$$\hat{\theta}(k) = \hat{\theta}(k-1) + K(k) \cdot \varphi(k) \cdot e(k) \qquad (2.33)$$

El esquema de identificación *[5]* puede representarse como se muestra en la ***Figura 2.2***, donde la función de mínimos cuadrados, a partir del regresor que se obtiene de las señales de entrada y salida de la planta, estima los parámetros de esta, cometiendo en la operación un error respecto a los reales que debe minimizarse.

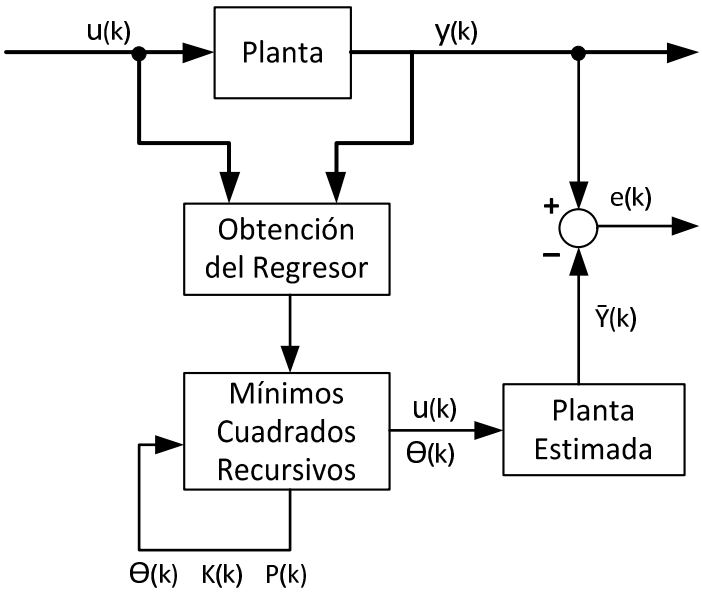

Figura 2.2. Esquema de identificación por Mínimos Cuadrados Recursivos

2.1.4. Mínimos Cuadrados con factor de olvido

El principal problema de los mínimos cuadrados recursivos *[9]* es la incapacidad para detectar cambios en parámetros, debido a que la matriz P se hace muy pequeña, teniéndose en cuenta en igual medida todas las muestras. Una manera de solucionarlo es introduciendo un coeficiente de olvido λ, de forma que se ponderan más las muestras recientes que las pasadas. Si se toma esta consideración, la nueva matriz de bondad de ajuste, viene dada por la siguiente expresión:

$$J = e^T(k) \cdot Q \cdot e(k) \qquad (2.34)$$

donde:

$$Q_k = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & \lambda & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda^{k-N} \end{bmatrix} \qquad \textbf{(2.35)}$$

Realizándose nuevamente los cálculos descritos con anterioridad, se puede llegar a las expresiones siguientes:

$$\hat{\theta}(k) = \hat{\theta}(k-1) - P(k) \cdot \varphi(k) \cdot e(k) \qquad \textbf{(2.36)}$$

$$P(k) = \frac{1}{\lambda}\left[P(k-1) - \frac{P(k-1) \cdot \varphi(k) \cdot \varphi^T(k) \cdot P(k-1)}{\lambda + \varphi^T(k) \cdot P(k-1) \cdot \varphi(k)}\right] \qquad \textbf{(2.37)}$$

o lo que es lo mismo:

$$P(k) = \frac{1}{\lambda}\left[P(k-1) - \frac{P(k-1) \cdot \varphi(k) \cdot \varphi^T(k) \cdot P(k-1)}{\lambda + \varphi^T(k) \cdot P(k-1) \cdot \varphi(k)}\right] \qquad \textbf{(2.38)}$$

$$K(k) = \frac{P(k-1)}{\lambda + \varphi^T(k) \cdot P(k-1) \cdot \varphi(k)} \qquad \textbf{(2.39)}$$

$$e(k) = y(k) - \varphi^T(k) \cdot \hat{\theta}(k-1) \qquad \textbf{(2.40)}$$

$$\hat{\theta}(k) = \hat{\theta}(k-1) + K(k) \cdot \varphi(k) \cdot e(k) \qquad \textbf{(2.41)}$$

El valor de λ debe esta comprendido entre 0 y 1, y en función de este valor se pueden observar los siguientes comportamientos *[3]*:

- λ ***pequeño –*** Cuanto más pequeño sea antes se descartan datos pasados debido a que las matrices P y K se hacen más grandes.
- λ ***grande –*** Para valores cercanos a la unidad se producen problemas a la hora de detectar cambios. Si este parámetro toma el valor de 1, el método se comporta como unos mínimos cuadrados sin factor de olvido.

Por norma general este parámetro se suele tomar entre 0.9 y 0.98, aunque depende bastante de los parámetros asociados del sistema.

2.2. Control Adaptativo

Los controladores convencionales están pensados para controlar sistemas dinámicos cuyos parámetros no cambian, es decir, son invariantes con el tiempo o, como ocurre en la realidad, sistemas sin grandes perturbaciones cuyos parámetros no varían excesivamente al trabajar en torno a un punto de funcionamiento.

Sin embargo es muy normal encontrarse con sistemas cuya dinámica cambia de forma no lineal en un instante, como es el ejemplo de un motor de corriente continua que varía sus parámetros en función de la carga. En estos casos el uso de controladores convencionales hace que el sistema no se comporte como se pretende en todas las situaciones.

Como solución a este problema surge el Control Adaptativo *[23]*, cuya finalidad es ir calculando los parámetros del regulador en función de la dinámica que el sistema tenga en cada instante de tiempo con el fin de conseguir que el comportamiento en lazo cerrado sea siempre el mismo.

Para conseguirlo se utiliza en reguladores STR (Self-Tunning Regulators) un esquema similar al de la ***Figura 2.3***, donde se parte del lazo típico de control en bucle cerrado al que se le añade un segundo lazo encargado de la identificación y cálculo de los parámetros del regulador adaptativo. Así en cada muestra de tiempo se lleva a cabo una identificación de la planta que sirve de base para la obtención de los valores del regulador, el cual ajusta la dinámica del sistema haciendo uso de los nuevos valores en la siguiente muestra de tiempo.

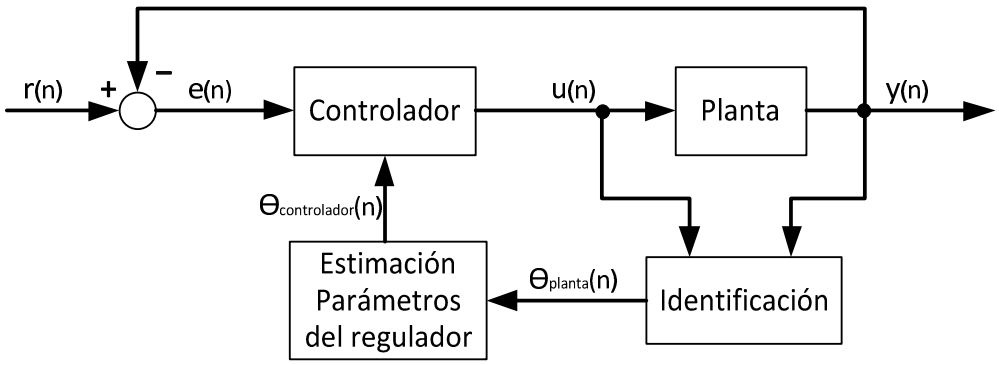

Figura 2.3. Esquema general del control adaptativo

Para ello se debe tener acceso a la entrada de la planta ($u(n)$) y la respuesta del sistema ante esta entrada ($y(n)$), con el fin de poder realizarse la identificación de la que se obtienen sus parámetros estimados ($\theta_{plata}(n)$), haciendo uso de algún método como puede ser el de mínimos cuadrados recursivo. A partir de estos valores se calculan los parámetros que requiere el controlador implementado ($\theta_{controlador}(n)$) y se le vuelven a pasar para que la dinámica se ajuste en el instante ($n + 1$) a la deseada en función de lo obtenido en el instante

(n). Este proceso se repite en cada uno de los tiempos de muestreo por lo que cualquier cambio que sufra la planta es tenido en cuenta reajustando el controlador para obtenerse el comportamiento deseado del sistema en su conjunto.

Aunque a primera vista la idea del control adaptativo puede resultar muy atractiva, conlleva una serie de inconvenientes que deben hacer que se replantee la necesidad de su uso, como que el ajuste es más complicado que el de un PID clásico o que hay que encontrar un método de ajuste del regulador sin conocer realmente la dinámica del sistema. De esta forma es interesante ver claramente cuando es más ventajoso el uso de controladores adaptativos y cuando es preferible la utilización de controladores convencionales.

2.2.1. Regulador PID

A pesar de que la aparición de estos controladores data de principios del siglo XX y del avance en las técnicas de control *[26]*, hoy en día es uno de los mecanismos de control realimentado más utilizados en la industria, sobre todo en procesos térmicos y químicos, quizás debido al gran esfuerzo realizado en el desarrollo de nuevos algoritmos de control basados en estructuras PID *[6].*

Este regulador depende de tres parámetros distintos:

- ***Proporcional P –*** Depende del error actual y modifica la salida en proporción de este.

$$P = K_p e(t) \tag{2.42}$$

- ***Integral I –*** Depende de la suma de errores pasados.

$$I = K_i \int_0^t e(\tau)\, d\tau \tag{2.43}$$

- ***Derivativo D –*** Es la predicción de errores futuros. Calcula la variación del error mediante la pendiente del error en cada instante de tiempo.

$$D = K_d \frac{d}{dt} e(t) \tag{2.44}$$

De ahí que también se le conozca como regulador de tres términos. Aunque existen ciertas variantes en la implementación del regulador PID que solucionan algunos problemas que aparecen en este tipo de controlador, se puede decir de forma general que su esquema es el mostrado en la ***Figura 2.4***, donde la señal de control es el resultado obtenido de aplicar los términos proporcional, integral y derivativo a la señal de error, la cual representa la diferencia existente entre la referencia que se desea que siga el sistema y la señal real que se está obteniendo.

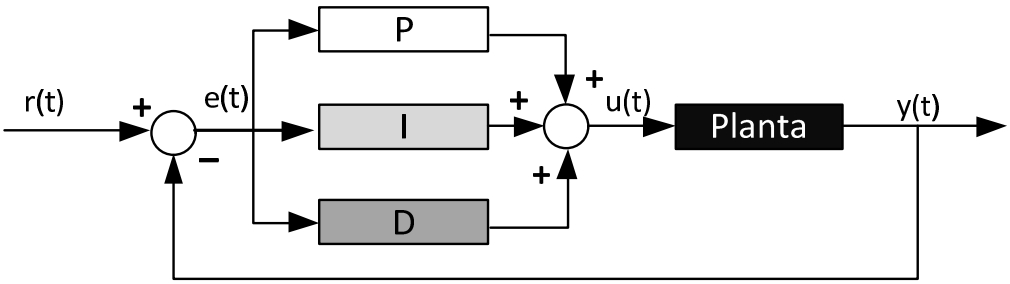

Figura 2.4. Esquema general de control de una planta con un regulador PID

Puesto que para realizar los cálculos de los que se obtienen los distintos parámetros del regulador se ha hecho uso de técnicas de asignación de polos *[7]*, es conveniente que su ecuación siga una expresión como la siguiente*:*

$$u(t) = \frac{r(t) \cdot (g_0 + g_1 + g_2) - (g_0 + g_1 \cdot z^{-1} + g_2 \cdot z^{-2}) \cdot y(t)}{1 - z^{-1}} \quad \textbf{(2.45)}$$

donde los términos que se definieron anteriormente, proporcional, integral y derivativo, se relacionan con g_0, g_1 y g_2 de la siguiente manera:

$$k_p = -g_1 - 2g_2 \quad \textbf{(2.46)}$$

$$k_d = g_2 \quad \textbf{(2.47)}$$

$$k_i = g_0 + g_1 + g_2 \quad \textbf{(2.48)}$$

Además se ha supuesto que la planta a regular se ajusta a la expresión:

$$y(t) = \frac{b_0 z^{-1}}{1 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}} u(t) \quad \textbf{(2.49)}$$

Con lo que, teniendo en cuenta las expresiones ***(2.45)*** y ***(2.49)***, la ecuación en lazo cerrado del conjunto es:

$$y(t) = \frac{b_0 z^{-1} \cdot (g_0 + g_1 + g_2)}{(1 - z^{-1})(1 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}) + b_0 z^{-1} \cdot (g_0 + g_1 \cdot z^{-1} + g_2 \cdot z^{-2})} r(t) \quad \textbf{(2.50)}$$

Lo que se desea ahora es encontrar los valores g_0, g_1 y g_2 que definen los términos del regulador para obtener la respuesta en lazo cerrado deseada. Para ello se hacen uso los criterios del tiempo de establecimiento y la frecuencia natural amortiguada.

Si lo que se quiere es que el sistema se comporte como un sistema de segundo orden, la expresión del polinomio característico debe tener la forma:

$$P = 1 + t_1 z^{-1} + t_2 z^{-2} \quad \textit{(2.51)}$$

donde t_1 y t_2 son:

$$t_1 = -2e^{-\xi \omega_n T cos(T\omega_n \sqrt{(1-\xi^2)})} \quad \textit{(2.52)}$$

$$t_2 = e^{-2\xi \omega_n T} \quad \textit{(2.53)}$$

siendo ξ el factor de amortiguamiento, ω_n la frecuencia natural y T el periodo de muestreo del sistema.

Así, el polinomio característico de la expresión ***(2.50)*** debe ser igual a la expresión ***(2.51)*** pues ambos deben tener los mismos ceros:

$$(1 - z^{-1})(1 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}) + b_0 z^{-1} \cdot (g_0 + g_1 \cdot z^{-1} + g_2 \cdot z^{-2}) = 1 + t_1 z^{-1} + t_2 z^{-2} \quad \textit{(2.54)}$$

Con lo que se pueden despejar los parámetros g_0, g_1 y g_2 que se quieren hallar, definiéndose con ello los tres términos del regulador:

$$g_0 = \frac{t_1 + (1 - a_1)}{b_0} \quad \textit{(2.55)}$$

$$g_1 = \frac{t_2 + (a_1 - a_2)}{b_0} \quad \textit{(2.56)}$$

$$g_2 = \frac{a_2}{b_0} \quad \textit{(2.57)}$$

Si en lugar de un sistema de segundo orden, la elección fuese uno de primero, el polinomio característico debe tener la siguiente forma:

$$P = 1 + t_1 z^{-1} \quad \textit{(2.58)}$$

donde:

$$t_1 = e^{-\frac{T}{\beta}} \quad \textit{(2.59)}$$

siendo β la constante de tiempo deseada para el sistema en lazo cerrado y T el intervalo de muestreo.

Nuevamente el polinomio característico de la expresión ***(2.50)*** debe ser igual a la expresión ***(2.58)***, obteniéndose para este caso:

$$(1 - z^{-1})(1 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}) + b_0 z^{-1} \cdot (g_0 + g_1 \cdot z^{-1} + g_2 \cdot z^{-2}) = 1 + t_1 z^{-1} \quad \textit{(2.60)}$$

Con lo que al despejar se obtienen los parámetros g_0, g_1 y g_2, cuyas expresiones están definidas por:

$$g_0 = \frac{t_1 + (1 - a_1)}{b_0} \quad \textit{(2.61)}$$

$$g_1 = \frac{a_1 - a_2}{b_0} \quad \textit{(2.62)}$$

$$g_2 = \frac{a_2}{b_0} \quad \textit{(2.63)}$$

El esquema de control que se obtiene finalmente para un sistema tanto de primer como de segundo orden es el que se muestra en la ***Figura 2.5***, el cual representa la expresión ***(2.45)***.

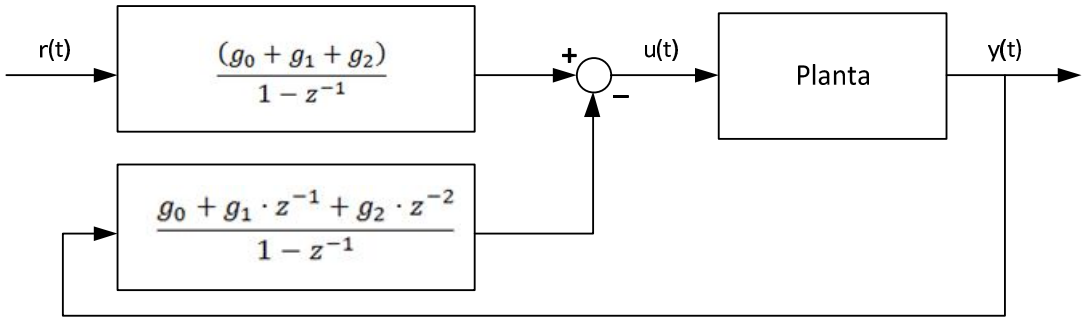

Figura 2.5. Esquema de control de un regulador PID real

Se puede ver que los parámetros del regulador dependen de los parámetros del sistema, además del tiempo de muestreo, lo que implica que tenemos que conocer el sistema para poder calcular el regulador. Sin embargo cuando se trata de control adaptativo esto no pasa en los primeros periodos, al iniciar el sistema su arranque, pues aún no se ha comenzado la identificación, teniéndose que encontrar soluciones que eviten comportamientos indeseados que puedan llevar al sistema a inestabilidades.

2.2.2. Controlador de tiempo de establecimiento finito (Dead-Beat)

El control Dead-Beat se caracteriza por establecer un error cero en un número finito de pasos discretos. Para ello se reemplazan los polos de la planta y se impone que los polos del sistema completo estén en el origen. Debido al hecho del reemplazo de polos de la planta, se debe tener cuidado con aquellos que estén en las cercanías del círculo unidad.

Además, puesto que el control requiere una respuesta Dead-Beat sólo en los instantes de muestreo *[8]*, pueden producirse oscilaciones que no decrezcan en la respuesta permanente entre los instantes de muestreo, incluso siendo el sistema de control internamente estable.

Se supone que la forma que tiene la planta que se quiere controlar *[7]* sigue la expresión:

$$G(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_n z^{-n}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}} \qquad \textbf{(2.64)}$$

y que la ecuación de transferencia de un controlador de tiempo finito es:

$$G_{db}(z) = \frac{p_0 + p_1 z^{-1} + p_2 z^{-2} + \cdots + p_n z^{-n}}{q_0 + q_1 z^{-1} + q_2 z^{-2} + \cdots + q_n z^{-n}} \qquad \textbf{(2.65)}$$

donde los parámetros p_i y q_i están relacionados con los parámetros de la planta y responden a las siguientes expresiones:

$$p_i \Longrightarrow \begin{cases} p_0 = \dfrac{r}{\sum b_i} = \dfrac{r}{b_0 + b_1 + b_2 + \cdots + b_n} \\ p_1 = a_1 \cdot p_0 \\ p_2 = a_2 \cdot p_0 \\ \vdots \\ p_n = a_n \cdot p_0 \end{cases} \qquad \textbf{(2.66)}$$

$$q_i \Longrightarrow \begin{cases} q_0 = r - p_0 \cdot b_0 \\ q_1 = -p_0 \cdot b_1 \\ q_2 = -p_0 \cdot b_2 \\ \vdots \\ q_n = -p_0 \cdot b_n \end{cases} \qquad \textbf{(2.67)}$$

Debido a que los coeficientes del controlador dependen únicamente de los parámetros del sistema y de la entrada de referencia, es uno de los controladores más sencillos de implementar de forma adaptativa, sin embargo vuelve a aparecer el problema del regulador PID y es que no se tiene información del sistema antes del instante inicial.

El esquema de control que se obtiene se muestra en la ***Figura 2.6*** y responde a la ecuaciones expresadas en ***(2.65)*** y ***(2.64)***.

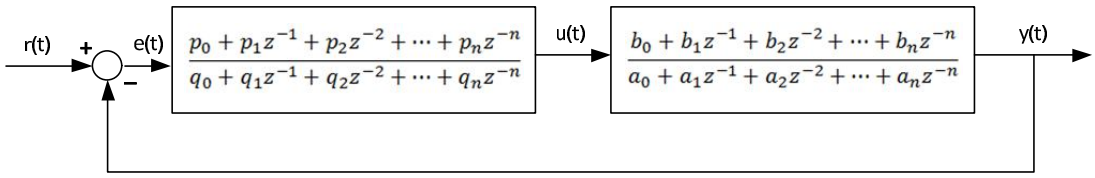

Figura 2.6. Esquema de control de un controlador Dead-Beat real

Hay que tener en cuenta varios puntos *[8]* al hacerse uso de este regulador:

1. Puesto que se obtienen polos múltiples en lazo cerrado en el origen, este controlador es muy sensible a variaciones en los parámetros del sistema.
2. Aunque este controlador está diseñado para obtener una respuesta de error cero en un número determinado de muestras, pueden mostrarse respuestas transitorias inferiores en función del tipo de entrada. Solo se tendrá un control óptimo para el tipo de entrada para el que se diseñó.
3. En un principio el periodo de muestreo no afecta para el cálculo de los términos del regulador, sin embargo, variaciones en el periodo de muestreo pueden provocar modificaciones en la dinámica del sistema e incluso la inestabilidad del mismo.
4. Como ya se mencionó, debido a que se requiere una respuesta Dead-Beat sólo en los instantes de muestreo, pueden producirse oscilaciones que no decrezcan en la respuesta permanente entre dichos instantes, incluso siendo el sistema de control internamente estable.

Se observa que la implementación de los reguladores paramétricos que se han propuestos no implican una gran dificultad desde el punto de vista matemático, dependiendo los valores de estos enteramente de los parámetros de la planta, de forma que la posibilidad de ser implementados y utilizados en plantas físicas, dependen más de las herramientas usadas y de los resultados obtenidos en entornos reales.

Capítulo 3. Control con Simulink en External Mode

Simulink ofrece las herramientas necesarias para implementar controladores adaptativos capaces de operar en tiempo real con plantas físicas, además el uso de *External Mode* permite tiempos de muestreo bajos, interesantes para sistemas con dinámicas rápidas. Todo ello haciendo uso de una interfaz gráfica, que simplifica el proceso de implementación, para la creación de modelos de bloques que pueden ser posteriormente exportados a otros lenguajes de programación como C, muy utilizado en aplicaciones industriales.

El uso de funciones definidas por el usuario, incluidas en los modelos de Simulink, permite la utilización de diversos lenguajes y funciones especializadas, entre las que se encuentran las propias de MATLAB, pudiéndose aprovechar todo el potencial de cálculo que estas nos brindan sin la necesidad de tener que partir desde cero para su creación, de una manera rápida y sencilla.

3.1. MATLAB

MATLAB *[10]* es un entorno informático con un lenguaje de alto nivel que permite programar tareas de cálculo más rápidamente que otros lenguajes al no tener que hacer tareas administrativas de bajo nivel, como es el caso de la declaración de variables. Integra técnicas de cálculo, adquisición, exploración y visualización en un entorno de programación fácil de usar, donde problemas y soluciones se expresan en una notación matemática familiar a través del uso de matrices, de ahí su nombre, MATrix LABoratory (Laboratorio de matrices).

Las características principales que presenta son:

- Lenguaje de alto nivel.
- Entorno de desarrollo para manejo de código, archivos y datos.
- Herramientas interactivas para la exploración iterativa, el diseño y la resolución de problemas.
- Funciones matemáticas para álgebra lineal, estadística, análisis de Fourier, filtrado, optimización e integración numérica.
- Funciones 2D y 3D para la visualización gráfica de datos.
- Herramientas que permiten la construcción de interfaces gráficas por el usuario.
- Funciones con el fin de alcanzar una integración de los algoritmos de MATLAB con otros lenguajes y aplicaciones externas, tales como C, C++, Fortran, Java, COM y Microsoft Excel.

MATLAB proporciona una interfaz de línea de comandos de forma que a través de funciones escritas o con M-file scripts se puede tener el control total del programa que se ha creado para Real-Time Windows o añadirles funcionalidades en tiempo de ejecución, ya sea en *Normal Mode* o *External Mode*.

3.2. Simulink

Simulink *[11]* permite la simulación y diseño basado en modelos, siendo un entorno sencillo para simulación multidominio y diseño de sistemas dinámicos, y embebidos. Proporciona un entorno gráfico interactivo y un conjunto de librerías de bloques que permiten diseñar, simular, implementar y probar una gran variedad de sistemas variantes en el tiempo, incluyendo comunicaciones, control, procesamiento de señales, procesamiento de vídeo y procesamiento de imágenes.

Se disponen de bloques especialmente diseñados para Real-Time Windows Target y muchos otros existentes de uso general como los de la librería Simulink. Entre algunos de los bloques de las librerías Real-Time Windows Target se encuentran driver de entrada y salida con los que se puede interactuar con el medio físico, teniéndose la posibilidad de capturar y enviar información a través de tarjetas de adquisición de datos. Estos controladores de dispositivo están escritos en código C, por lo que es imprescindible un compilador C para su utilización en el modelo como un programa en *External Mode*.

También se necesita un sistema de archivos de destino que es el encargado de indicar para qué sistema es compilado el modelo y por lo tanto la plantilla Makefile necesaria para el uso de las distintas S-Function objeto. Este sistema de archivos de destino (System Target file) se debe elegir en las opciones de Real-Time Workshop. En el caso de trabajar en tiempo real en *External Mode* se hace uso de ***rtwin.tlc*** (Para Real-Time Windows Target).

Además de esto, Simulink permite desde la propia ventana ejecutar y controlar modelos creados y compilados con Real-Time Windows Target.

Como características principales de Simulink se tienen:

- Extensa biblioteca que puede ser ampliada con bloques predefinidos.
- Editor gráfico interactivo que permite la creación y gestión de los modelos de diagramas de bloques de una forma intuitiva.
- Capacidad de crear y gestionar diseños complejos mediante la división del modelo.
- Explorador del modelo que permite navegar, crear, configurar y buscar todas las señales, parámetros, propiedades y el código asociado al modelo.
- API's que permiten la interactuación con otros programas de simulación y la incorporación de código escrito a mano.
- Bloques de funciones para trasladar algoritmos de MATLAB a Simulink.
- Distintos modos de simulación.

- Debugger gráfico para examinar los resultados de la simulación y diagnosticar rendimientos y comportamientos indeseados en el diseño.
- Un completo acceso desde MATLAB para analizar y visualizar los resultados, crear cambios en el entorno del modelo, o definir señales y parámetros.
- Análisis del modelo y herramientas de diagnóstico para garantizar la coherencia e identificar posibles errores de modelado.

3.2.1. Real-Time Windows Target

Real-Time Windows Target *[12]* proporciona un motor que permite la ejecución en tiempo real de modelos de Simulink en equipos con Microsoft Windows, con una gran variedad de bloques que se conectan a una amplia gama de E/S de placas, permitiendo crear y controlar un sistema en tiempo real, cuya finalidad principal es la creación rápida de prototipos o de simulación HIL (hardware-in-the-loop).

Real-Time Windows es compatible con dos modos de simulación muy interesantes de Simulink, el primero es el *"Normal Mode"*, de sencilla operación, en tiempo real y con acceso a dispositivos E/S, aunque con ciertas limitaciones, y el segundo el *"External Mode"* proporciona un mayor rendimiento en tiempo real (con Simulink Coder™), pero realizar ciertas implementaciones resultan ser más complejas.

Sus características principales son:

- Ejecución de modelos en bule cerrado en tiempo real con Microsoft Windows.
- Posibilidad de visualización y modificación de ciertos parámetros mientras el modelo está en ejecución.
- El control de la ejecución es realizado directamente desde Simulink.
- Rendimiento en tiempo real superior a 500 Hz en modo de ejecución normal.
- Rendimiento en tiempo real superior a 5 kHz en modo de ejecución externo (con Simulink Coder).
- Con más de 250 módulos de E/S (incluyendo analógicas, digitales, contadores, codificadores y salidas de frecuencia) y protocolos de comunicación (UDP, serial y CAN).

A la hora de su utilización se debe crear un modelo y simularlo, haciéndose uso del software de Simulink en *Normal Mode*, pudiéndose añadir bloques de E/S de la librería Real-Time Windows Target. Posteriormente, en función de la actuación necesaria, se puede simular de nuevo en *Normal Mode* o cambiar a *External Mode*.

Los usos típicos de Real-Time Target Windows incluyen:

- ***Control en tiempo real.***
- ***Simulaciones en tiempo real HIL –*** Crear un prototipo de control conectado a una planta física o crear un prototipo de una planta conectada a un controlador real.

- ***Educación –*** Con el fin de enseñar los conceptos y procedimientos para modelar, simular y probar sistemas de tiempo real.

3.2.2. Simulink en "Normal Mode"

En *Normal Mode [13]*, Simulink ejecuta el algoritmo de simulación y utiliza una interfaz de comunicación para transferir datos hacia y desde las E/S de los controladores, que se ejecutan en un proceso separado en el núcleo de Windows como se puede ver en la ***Figura 3.1***.

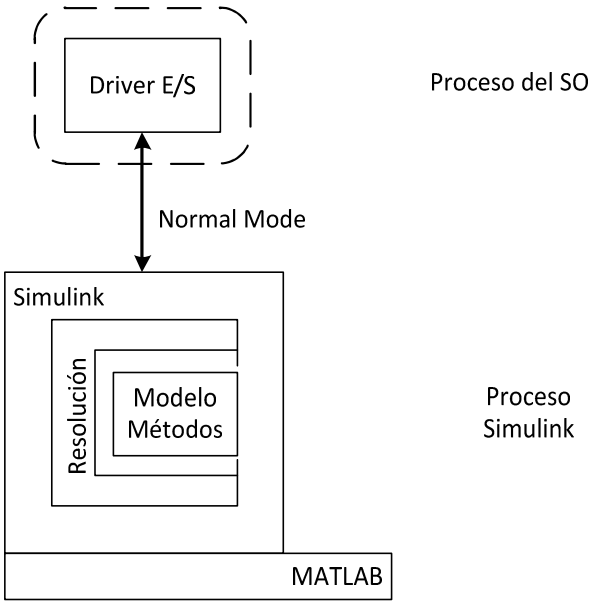

Figura 3.1. Ejecución de algoritmo en tiempo real en Normal Mode

Simulink, en *Normal Mode*, permite ser utilizado como interfaz gráfica para:

- ***Adquisición de la señal –*** Captura y muestra señales provenientes de la aplicación en tiempo real de ejecución.
- ***Ajuste de parámetros –*** Permite el cambio de valor de ciertos parámetros en el diagrama de bloques de Simulink, afectando estos cambios al modelo en tiempo real. Los efectos se propagan a través de los controladores de E/S hasta el hardware.

3.2.3. Simulink en "External Mode"

Para *External Mode [14]*, se debe generar código ejecutable con Simulink Coder y un compilador de C que permita ejecutar la aplicación de destino en tiempo real en el propio núcleo de Windows. Este modo actualmente no es compatible con instalaciones de 64 bits de Real-Time Windows.

External Mode requiere una interfaz de comunicaciones para poder pasar los parámetros externos al código ejecutable previamente compilado. El extremo receptor utiliza el mismo protocolo de comunicaciones para aceptar nuevos valores de los parámetros, que son insertados en la memoria para su uso por la aplicación en tiempo real.

Como se puede ver en la ***Figura 3.2***, Simulink solo se encarga de la gestión de una serie mínima de datos, mientras que tanto los controladores de las E/S como la ejecución de la aplicación obtenida se realizan en el núcleo de Windows. Esto tiene algunas ventajas claves como una mayor velocidad de ejecución, lo que trae aparejado la posibilidad de unos tiempos de muestreo menores que en el caso de *Normal Mode*, sin embargo es imprescindible que la aplicación se encuentre en un lenguaje comprensible para el sistema.

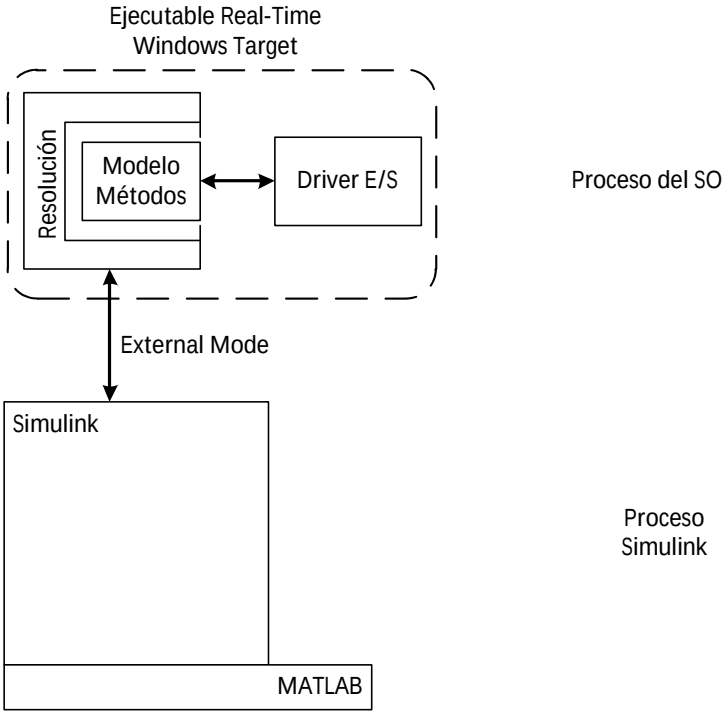

Figura 3.2. Ejecución de algoritmo en tiempo real en External Mode

La integración entre Simulink, trabajando en *External Mode* y el software de Real-Time Windows Target permite utilizar el modelo de Simulink como una interfaz gráfica de usuario:

- ***Adquisición de la señal –*** Permite capturar y mostrar las señales de la aplicación en tiempo real mientras se está ejecutando. Los datos de las señales se muestran o almacenan haciendo uso de los diversos bloques de Simulink. La mayoría de ellos se encuentran en la librería *Simulink/Skins*.
- ***Ajuste de parámetros –*** Se permite el cambio de ciertos parámetros en el diagrama de bloques de Simulink y que estos nuevos parámetros formen parte automáticamente de la aplicación en tiempo real que se está ejecutando, aunque en la mayoría de casos, cambios en los parámetros implica una nueva generación de código para que el modelo funcione.

3.3. Funciones definidas por el usuario

Las funciones definidas por el usuario son una serie de bloques disponibles en la librería *Simulink/User-Defined Functions* que permiten aumentar la funcionalidad del modelo creado, gracias a que posibilitan trasladar funciones y código perteneciente de MATLAB y otros lenguajes al modelo en Simulink, de forma escrita y sencilla.

3.3.1. Fcn

El bloque Fcn, que se muestra en la ***Figura 3.3*** donde se pueden ver sus parámetros de configuración, aplica una expresión matemática definida por el usuario que es función de la entrada *[15]*.

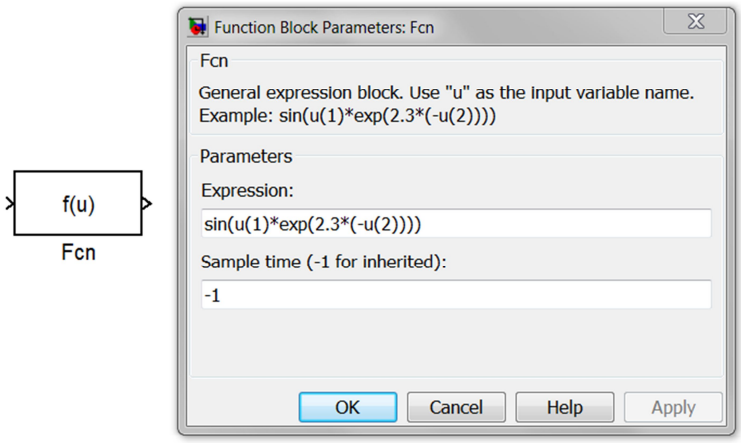

Figura 3.3. Bloque de la función definida por el usuario Fcn

La expresión difiere respecto a una expresión de MATLAB en que esta no puede realizar cálculos de matrices. Además de no ser compatible con el operador dos puntos (:). Así, la entrada a un Bloque Fcn puede ser un escalar o un vector, sin embargo, la salida es siempre un escalar. Si lo que se desea es obtener un vector de salida se puede utilizar un Function block para ello.

Se permite hacer uso en la expresión de:

- Constantes numéricas.
- Operadores aritméticos (+ , - , * , / , ^).
- Operadores relacionales (== , => , <> , =< , =!). La expresión devuelve 1 si la relación es verdadera, de lo contrario, devuelve 0.
- Operadores lógicos (! , && , | |). La expresión devuelve 1 si la relación es verdadera, de lo contrario, devuelve 0.
- Paréntesis.
- Funciones matemáticas (abs, acos, asin, atan, atan2, ceil, cos, cosh, exp, fabs, floor, hypot, ln, log, log10, pow, power, rem, sgn, sin, sinh, sqrt, tan y tanh).
- Las variables del Workspace.

3.3.2. MATLAB Function

Con un bloque MATLAB Function *[15]* se puede escribir una función de MATLAB para ser usada en un modelo de Simulink. La función de MATLAB creada es ejecutada durante la simulación y genera código para Simulink.

Al contrario de lo que pasaba con las Fcn, ahora se puede especificar el número de entradas y salidas necesarias para implementar la función de MATLAB en el encabezado de esta, como argumentos y valores devueltos según se puede observar en la ***Figura 3.4***, además se tiene la posibilidad de que tanto las entradas como las salidas sean en forma matricial.

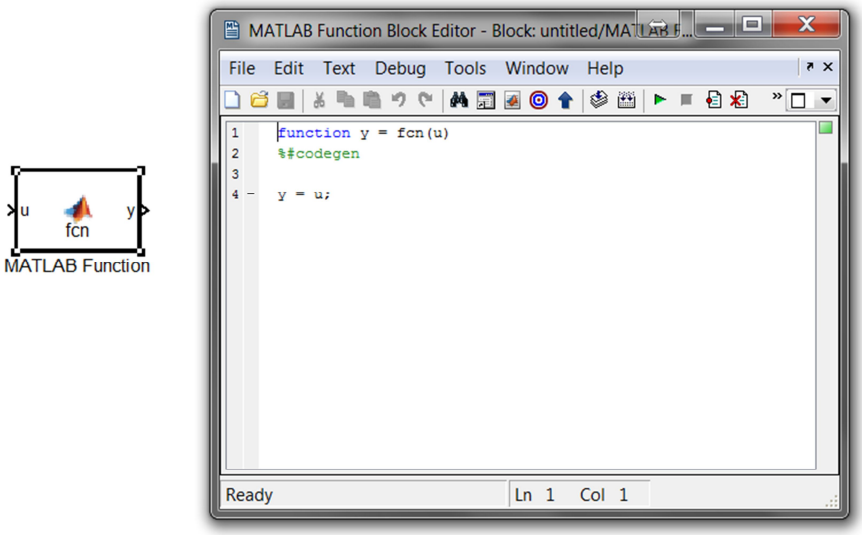

Figura 3.4. Bloque de la función definida por el usuario MATLAB Function

Este bloque puede generar un código embebido eficiente, que se basa en el análisis del tamaño, clase y complejidad de cada variable. Sin embargo este análisis impone ciertas restricciones:

- La primera asignación de una variable debe definir su tamaño, clase y complejidad.
- Por lo general, no se pueden reasignar propiedades a las variables después de la asignación inicial, excepto cuando se utilizan datos de tamaños variables o se reutilizan las variables en el código para diferentes propósitos.

Además de estas restricciones de tipo lingüísticas, este bloque de MATLAB sólo admite un subconjunto de las funciones de las que dispone. Estas se incluyen en categorías comunes tales como:

- Funciones aritméticas como plus, minus y power.
- Operaciones con matrices como size y length.
- Operaciones avanzadas de matrices como lu, inv, svd y chol.
- Funciones trigonométricas como sin, cos, sinh y cosh.

Aunque el código de este bloque trata de producir exactamente los mismos resultados que se obtendrían con MATLAB, las diferencias pueden deberse a errores de redondeo, que pueden ser de unos pocos eps inicialmente para ir aumentando con el paso del tiempo tras realizarse un gran número de operaciones repetidas. Esta variable eps, es la mínima distancia que hay entre la representación de un número y el siguiente más cercano, y surge debido a la naturaleza discreta de los sistemas computacionales, con lo que en esencia el eps es el mínimo error de precisión que se está cometiendo. Además la confianza en el comportamiento de los NaN (Not a Number), por ejemplo, no es recomendable. Todo esto se une al hecho de que diferentes compiladores de C pueden producir resultados distintos para el mismo cálculo.

Las llamadas recursivas no están permitidas en los bloques de funciones de MATLAB.

3.3.3. S-Function

Una S-Function (System-Functions) *[15]* es un lenguaje descriptivo escrito dentro de un bloque de Simulink en MATLAB, C, C++ o Fortran. Para C, C++ y Fortran las S-Function se compilan como archivos MEX haciendo uso de mex utility. Al igual que ocurre con otros archivos MEX, las S-Function se vinculan dinámicamente a subrutinas que el intérprete de MATLAB puede cargar automáticamente y ejecutar. En la ventana de configuración que se muestra en la ***Figura 3.5*** se puede ver que se necesitan unos parámetros de base para crear la S-Función, que se pueden introducir a través de una máscara, y el archivo que contenga el código de esta.

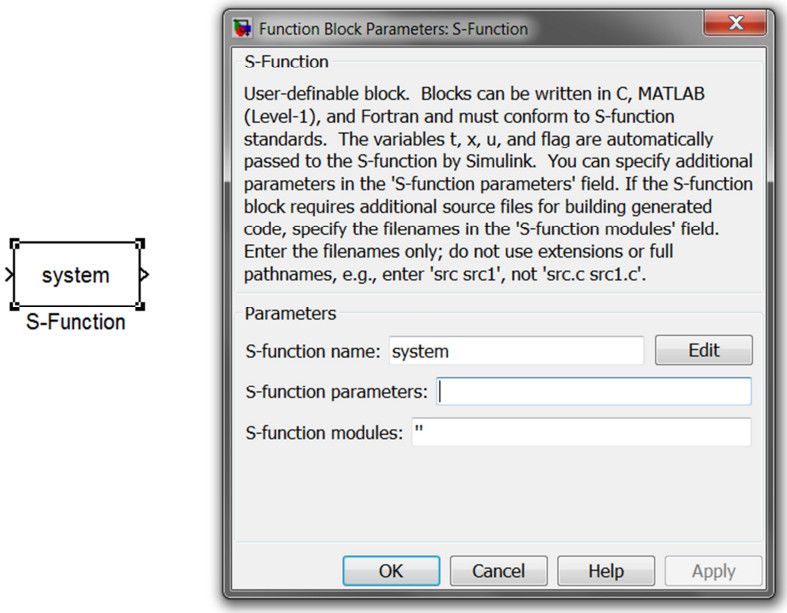

Figura 3.5. Bloque de la función definida por el usuario S-Function

Las S-Function utiliza una sintaxis de llamada especial denominada S-Function API, que le permite interactuar con el motor de Simulink. Esta interacción es muy similar a la interacción que tiene lugar entre el motor y el sistema de bloques de Simulink.

Las S-Function son un poderoso mecanismo que permite ampliar las capacidades de Simulink, pudiéndose integrar estas funciones en sistemas continuos, discretos e híbridos. Siguiéndose una serie de pautas simples se puede implementar un algoritmo haciéndose uso de una S-Funtion que puede posteriormente ser usada como un bloque en un modelo distinto de Simulink.

En todo momento se tiene la posibilidad de personalizar el código generado para las S-Function escribiéndose el archivo TLC (Target Language Compiler) y está permitido el uso de máscaras para crear cuadros de diálogo personalizados e iconos para el bloque de la S-Function. Los cuadros de diálogo de las máscaras pueden hacer que sea más fácil especificar parámetros adicionales.

Las S-Function son de gran utilidad para un amplio número de aplicaciones, entre las que se incluyen:

- Creación de nuevos bloques de propósito general.
- Añadir bloques que representen controladores de dispositivos físicos.
- Incorporar en la simulación, código ya existente de C.
- Describir un sistema como un conjunto de ecuaciones matemáticas.
- Uso de animaciones gráficas.

El uso más común de las S-Function es la primera, crear bloques de Simulink que se puedan usar muchas veces, en uno o varios modelos, modificando únicamente los parámetros con cada instancia del bloque.

Para crear las S-function es necesario entender cómo trabajan este tipo de funciones. Tal conocimiento requiere una comprensión de cómo el motor de Simulink simula un modelo, incluyendo las matemáticas de los bloques.

Un bloque de Simulink se compone de un conjunto de entradas, estados y salidas, como se puede ver en la ***Figura 3.6***, donde las salidas son función del tiempo de simulación, las entradas y los estados.

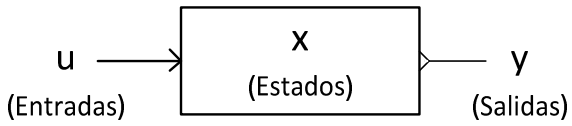

Figura 3.6. Diagrama de la relación de entradas, estados y salidas en un modelo de Simulink

De esta forma se puede representar las relaciones matemáticas entre las entradas, salidas, estados, y el tiempo de simulación con las siguientes expresiones:

$$\begin{cases} y = f_o(t, x, u) & (Output) \\ \dot{x} = f_d(t, x, u) & (Derivatives) \\ x_{d_{k+1}} = f_u(t, x_c, x_d, u) & (Update) \end{cases} \qquad (3.1)$$

donde:

$$x = [x_c; x_d] \qquad (3.2)$$

A partir de aquí la ejecución del modelo procede por etapas. En la ***Figura 3.7*** se ilustra las etapas de una simulación.

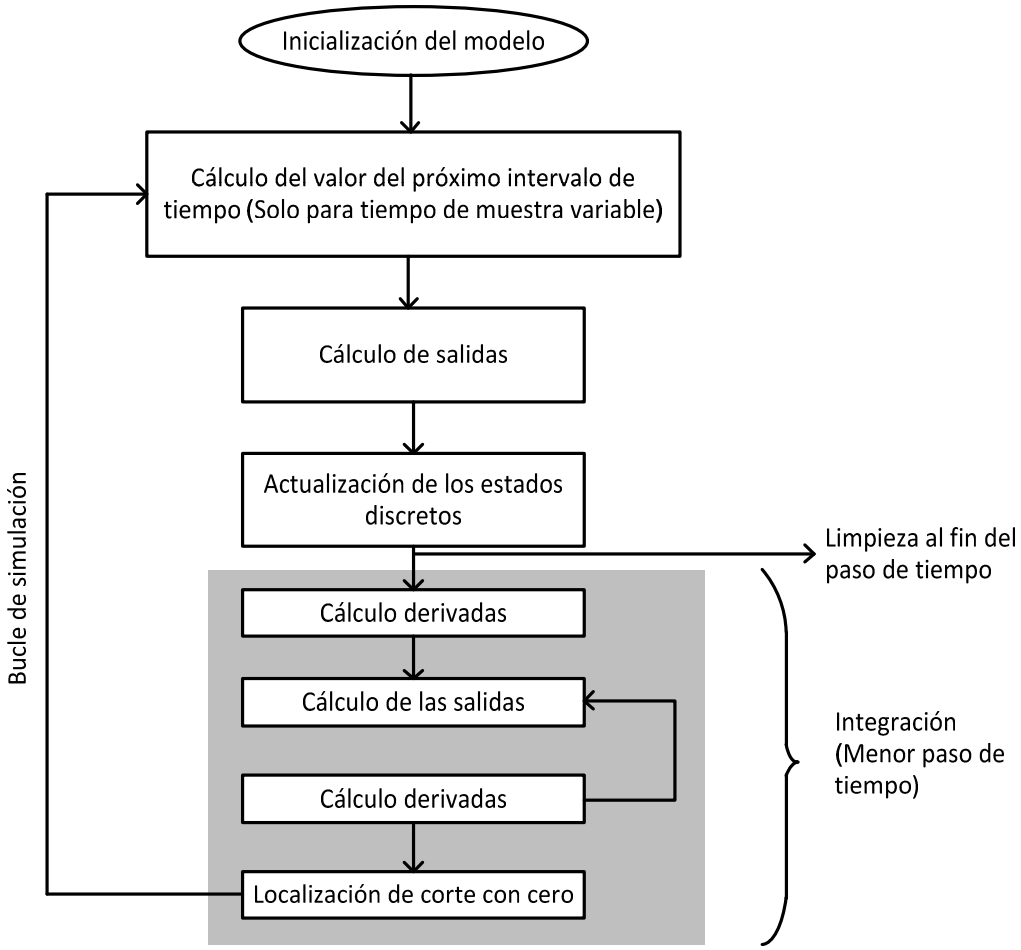

Figura 3.7. Etapas de ejecución de una S-Function

Durante la simulación de un modelo, en cada etapa, el motor de Simulink llama a los métodos apropiados para cada uno de los bloque de S-Function contenidos en él. Las tareas desempeñadas por las llamadas a métodos de las S-Function incluyen:

- ***Inicialización –*** Antes de la primera iteración se inicializa la S-Function:
  - Inicialización del SimStruct, una estructura de simulación que contiene información sobre la S-Function.
  - Ajuste del número y dimensiones de los puertos existentes tanto de entrada como de salida.
  - Configuración de los tiempos de muestreo del bloque.
  - Asignación de áreas de almacenamiento.

- ***Cálculo del siguiente paso de tiempo –*** En caso de haberse creado un bloque que haga uso de un paso de tiempo variable, esta etapa calcula el tiempo de la muestra siguiente.
- ***Cálculo de las salidas en el paso de tiempo actual –*** Después de completarse esta llamada, todos los puertos de salida del bloques son válidos para este paso de tiempo.
- ***Actualización de estados discretos en el paso de tiempo actual –*** en esta etapa el bloque realiza una única vez las instrucciones indicadas con la finalidad de actualizar los estados discretos.
- ***Integración –*** Solo se aplica a los modelos con estados continuos y/o no muestreados con raíces. Si la S-function contiene estados continuos, se calcula con el menor incremento de tiempo, además si tiene raíces no muestreadas, el motor también llama a las salidas y a los trozos de las raíces de la S-Function en el menor incremento de tiempo que le permita encontrar las raíces.

3.3.4. S-Function Builder

El bloque S-function Builder *[15]* crea una C MEX S-Function a partir de las especificaciones que se le definen en el constructor y el código fuente en C que se le proporcione. En la ***Figura 3.8*** se muestra el bloque de esta función definida por el usuario y su ventana de configuración.

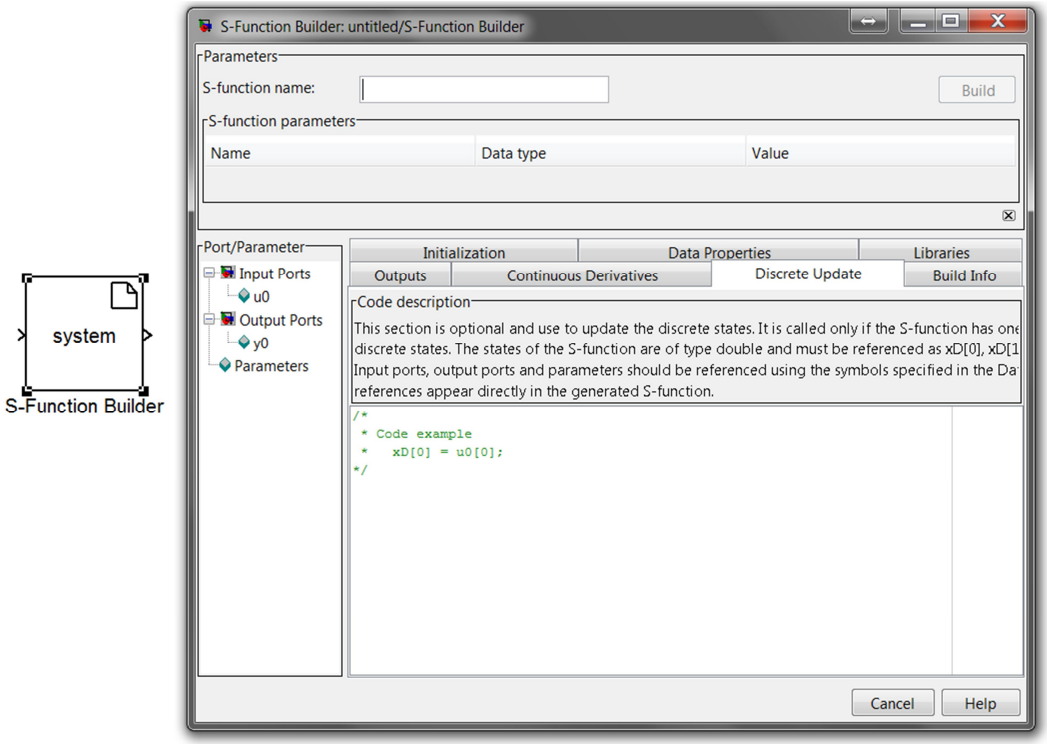

Figura 3.8. Bloque constructor de las S-Funciones, S-Function Builder

Las instancias del bloque S-Function Builder también sirven como contenedores para las S-Function generadas en los modelos de Simulink. Cuando se produce la simulación de un modelo que contiene las instancias de un bloque S-Function Builder, el software de Simulink invoca la S-Function generada asociada a cada instancia para calcular la salida de la instancia en cada paso de tiempo.

El bloque generador de S-Function no es compatible con el uso de enmascaramiento. Sin embargo sí que es posible usarse una máscara en un bloque de subsistema que contenga un bloque S-Function Builder.

El S-Function Builder construye la S-función básicamente en dos pasos *[17]*. En primer lugar, genera una serie de archivos de código fuente:

- ***sfun.c –*** donde SFUN es el nombre de la S-Function que se especificó en el campo de nombre del Builder. Este archivo contiene el código fuente en C que representa trozos estándar de la S-Function generada.
- ***sfun_wrapper.c –*** Contiene el código personalizado que se ha introducido en el cuadro de diálogo del S-Function Builder.
- ***sfun.tlc –*** Este archivo permite al S-function Builder ejecutarse en el entorno Simulink en *External Mode* o *Accelerator Mode*, además de permitirle procesos en línea durante la generación de código.
- ***sfun_bus.h –*** Si se especifica un puerto de entrada o de salida como un bus en el panel *Data Properties*, pero no especifica un archivo de cabecera, entonces el S-function Builder genera automáticamente este archivo de cabecera.

Después de generar el código fuente de la S-Function, el S-Function Builder utiliza el comando mex para construir el archivo MEX de la S-Function desde el código fuente generado, cualquier código fuente externo y las bibliotecas que se hayan especificado.

Aunque este constructor tiene sus limitaciones, como el uso de un lenguaje C con restricciones, presenta una ventaja fundamental y es que es capaz de generar de forma automática el archivo .tlc. En el apartado siguiente de este capítulo se explica que son estos archivos y para qué sirven, pero bajo el punto de vista más práctico, se puede resumir diciéndose que es el archivo que permite que las S-Function sean compiladas para ejecutarse, entre otros, en *External Mode*. Esto es básico, ya que se necesitan las S-Function para el control de sistemas, pues son capaces de manejar estados actuales y almacenar estados pasados. Aunque en principio pueda parecer que no es una ventaja reseñable, la realidad es que la creación de archivos .tlc no es algo trivial, siendo un tema muy complicado de gestionar de forma manual.

3.4. TLC

Los .tlc invocan al Target Language Compiler (TLC) desde la línea de comandos. El TLC *[19]* convierte el archivo de descripción del modelo, model.rtw (o archivos similares), en el código o texto específico de destino. Por lo general, no se llama a este comando porque el Simulink Coder, durante el proceso de construcción, lo invoca automáticamente cuando está generando el código.

De esta manera los TLC funciona con el software Simulink para generar el código *[20]* como se muestra en la ***Figura 3.9***.

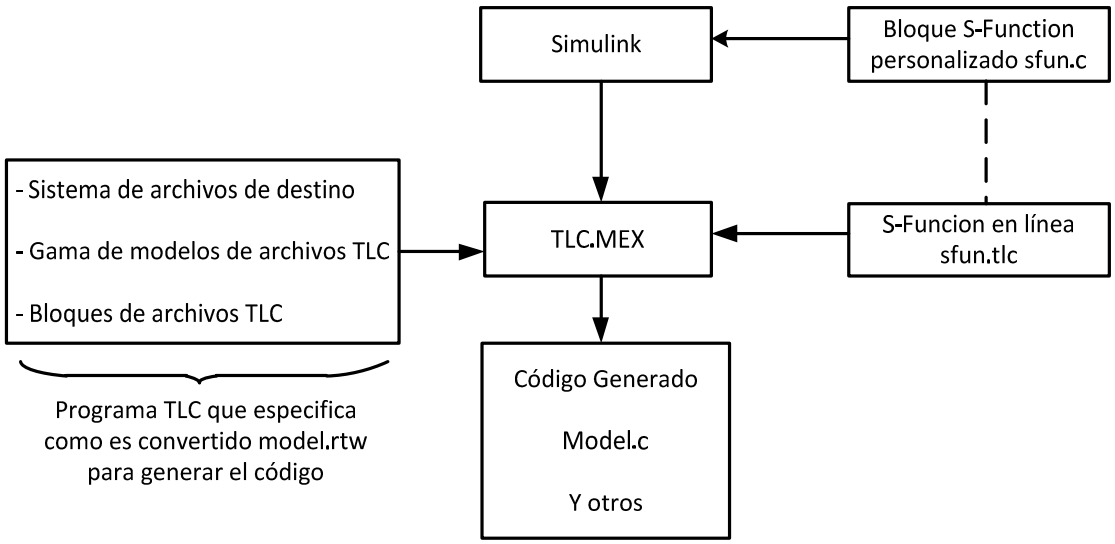

Figura 3.9. Pasos seguidos por TLC para la generación de código

Así como un programa C es una colección de archivos ASCII conectados con sentencias #include y archivos de objetos enlazados a una librería, un programa TLC es una colección de archivos ASCII también llamados scrips. Debido a que TLC es un lenguaje interpretado no hay archivos objetos. El único archivo objetivo que llama (con la directiva include%) es otro archivo objeto usado por el programa denominado punto de entrada.

Los archivos Target son el conjunto de archivos que son interpretados por TLC para transformar el código Simulink Coder (model.rtw) generado por Simulink en un código de destino específico. No se debe personalizar los archivos de TLC a pesar de que exista la posibilidad para hacerlo ya que estas modificaciones del TLC podrían no llegar a aplicarse durante el proceso de generación de código dando lugar a resultados impredecibles. Sólo se deberían personalizar los archivos TLC que hayan sido creados por uno mismo.

Estos archivos Target proporcionan la flexibilidad necesaria para personalizar el código generado por el compilador con el fin de adaptarse a unas necesidades específicas. Por ejemplo, si se utilizan un sistema de archivos de destino específico se puede producir código genérico de C o C++ desde el modelo de Simulink. Este código ejecutable no es específico de la plataforma.

MATLAB posee las herramientas necesarias y la potencia suficiente como para trabajar con controladores adaptativos con Real-Time Windows Target en *External Mode*. Su principal problemática radica en la dificultad de la creación de los tlc necesarios para el uso de S-Funciones, imprescindible para la gestión de estados discretos de valores pasados, y que se ve paliada en gran medida gracias a la incorporación de los builder incluidos a partir de la versión 7.9 de Simulink, contenida en MATLAB 2012a, capaces de generar estos archivos de forma automática.

Capítulo 4. Configuración y creación de los modelos implementados

La implementación de los controladores adaptativos se han llevado a cabo haciéndose uso de MATLAB 2012a, concretamente con la herramienta gráfica integrada de Simulink v7.9, que permite crear modelos de bloques y la generación automática de archivos .tlc, imprescindibles para la ejecución en *External Mode* de los bloques S-Funcion.

La instalación se ha realizado en un PC convencional con un sistema operativo Windows XP de 32 bits Servipack 3. La ejecución de modelos Simulink en Real-Time Windows Target requiere del sistema operativo Windows y el modo *External Mode* solo es compatible con sistemas de 32 bits en las versiones actuales de Simulink.

4.1. Configuración del sistema y comprobación del modelo

Para utilizarse una tarjeta de adquisición de datos que permita interactuar con el sistema externo real se debe usar un modelo de Simulink en *External Mode*. Ello requiere de una serie de pasos previos de configuración. Algunos de estos pasos, como se especifica en los mismos, son también imprescindibles para *Normal Mode*, modo necesario, como ya se comentó en el ***Apartado 3.2.1. Real-Time Windows Target***, para simular los modelos teóricos y comenzar a trabajar con los modelos prácticos que son utilizados en los ensayos de laboratorio. Los pasos a seguir son los siguientes:

1. Configurar el modo de resolución del modelo en el submenú "*Simulation / Configuration Parametres"* que se muestra en la ***Figura 4.1***. Se utiliza para todos los modelos del proyecto estados discretos con un paso de tiempo fijo, cuyo valor de tiempo depende del modelo y de la dinámica del sistema. Más adelante del presente capítulo, en la implementación de cada regulador, se especifica el paso de tiempo escogido y el motivo por el que se ha decidido elegir este paso. Esta primera configuración del modelo se debe realizar tanto si se trabaja en *External Mode* como en *Normal Mode*.

2. Se debe acceder a *Model Advisor* del menú Tools (*"Tools / Model Adisor"*), el cual se muestra en la ***Figura 4.2***, y que permite ejecutar en modo de prueba un modelo y generar un reporte con los parámetros de configuración, así como un diagnóstico de eficiencia en sus tareas. Este paso para la comprobación del modelo es recomendable tanto si se trabaja en *External Mode* como en *Normal Mode*. Este punto no aporta nada en la configuración, pero indica si el modelo realizado es correcto para ser ejecutado.

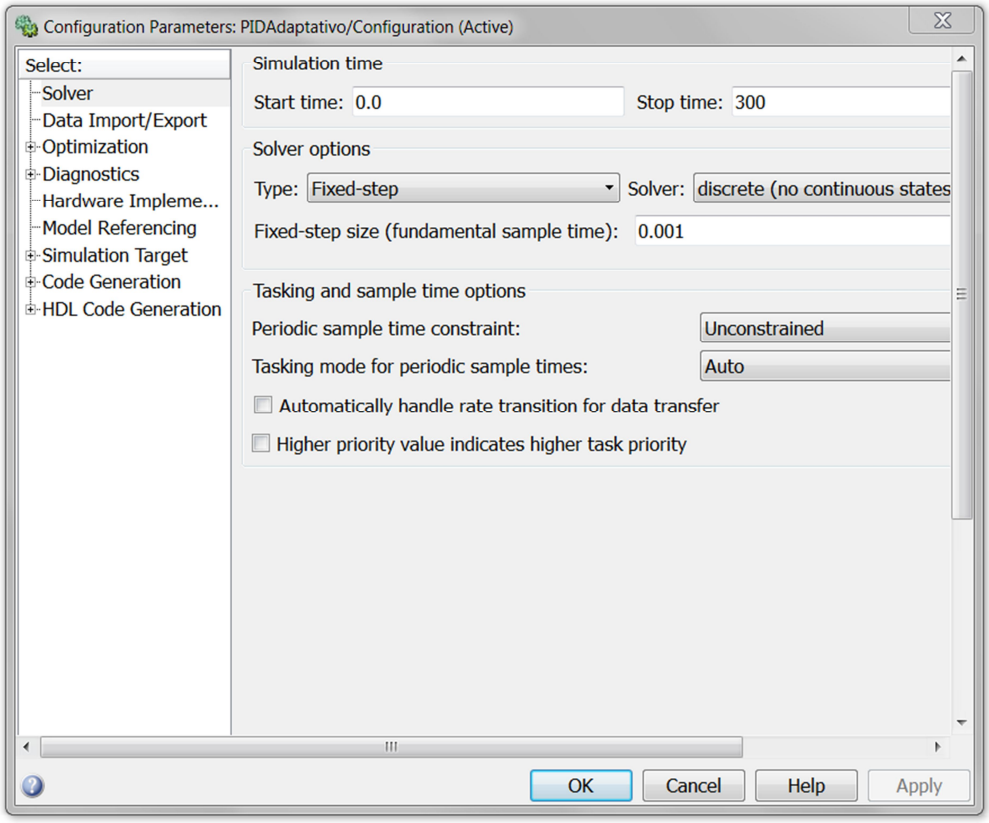

Figura 4.1. Submenú "Solver" de la configuración del modelo "Simulation / Configuration Parametres"

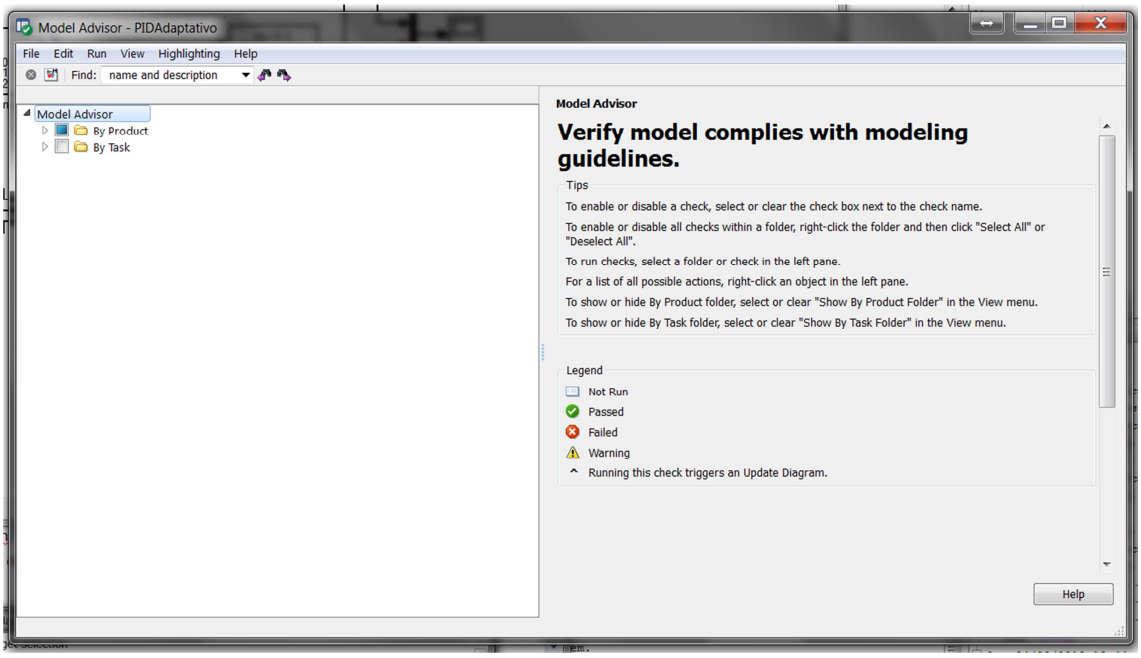

Figura 4.2. Panel principal de "Model Advisor"

3. Elegir el sistema de archivos de destino como *Real-Time Windows Target*. Para ello hay que acceder a *"Simulation / Configuration Parametres"* y en la pestaña *"Code generation"*, del menú que aparece, configurar la opción *"System target file"* como "*rtwin.tlc"* tal y como se observa en la ***Figura 4.3***.

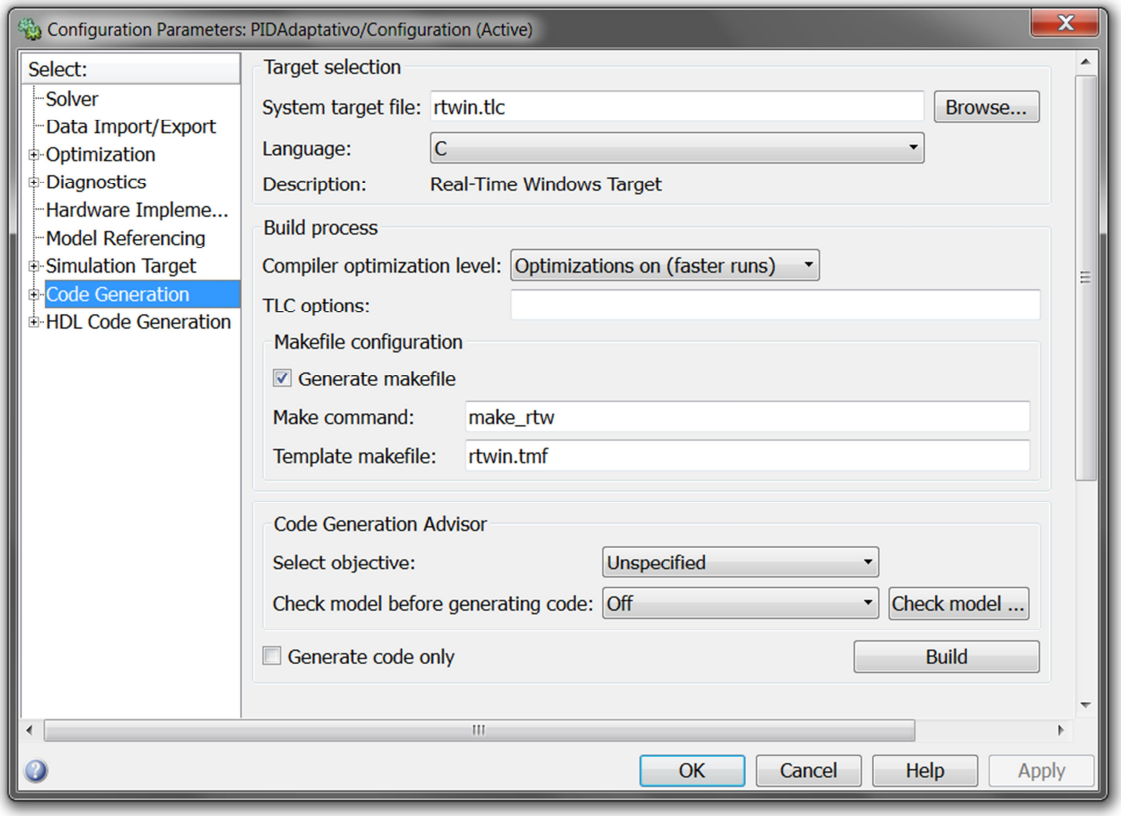

Figura 4.3. Submenú "Code Generation" de la configuración del modelo "Simulation / Configuration Parametres"

4. Compilar el modelo generando el código C necesario para ser ejecutado en *External Mode*. Para ello se puede hacer uso del botón incremental build de la ***Figura 4.4***

Figura 4.4. Botón Incremental Build

que realiza un compilado incremental de todas aquellas partes que no hayan sido todavía compiladas o hayan sido modificadas desde la última compilación, o también existe la posibilidad de compilar el proyecto completo. El problema de la compilación completa es que requiere de más tiempo y no es excesivamente práctico usar esta opción cuando se está probando y cambiando elementos del modelo de forma frecuente.

Es posible que al intentar compilar aparezca un error relacionado con el Mex -file, sobre todo si es la primera vez que se compila y aún no se ha configurado. Para solucionarlo solo hay que seguir los pasos que se muestran en el error, aunque esencialmente es usar el comando *mex –setup* para configurar el compilador. Mex tiene la función de compilar y

linkar archivos de código fuente en una biblioteca compartida llamada Mex -file, que son ejecutables desde MATLAB. El archivo resultante depende de la extensión de la plataforma.

5. Conectar el modelo pulsando el botón *Conect* mostrado en la ***Figura 4.5.***

Figura 4.5. Botón Conect

En este punto puede aparecer un error debido al Real-Time Windows Target, lo que no permite ejecutar modelos en tiempo real. La solución pasa por hacer uso del comando *rtwintgt –setup* que descarga e instala el paquete necesario que permite la ejecución de código C generado por Simulink Coder en un PC en tiempo real. Si aun así no funcionase puede ser debido a que el sistema operativo utilizado es Windows de 64 bits y MATLAB aún no da soporte para estos sistemas. La única solución posible es instalar MATLAB en un sistema de 32 bits bajo un sistema operativo de Windows.

6. Iniciar la ejecución en tiempo real. Una vez configurado el modelo y comprobado que es correcto, ya se puede ejecutar en tiempo real haciendo uso del botón Start Real-Time Code mostrado en la ***Figura 4.6.***

Figura 4.6. Botón Start Real-Time Code

4.2. Identificación de procesos

Como ya se ha hablado a lo largo del proyecto es imprescindible la identificación de la planta con el fin de obtener sus parámetros y así poder calcularse a posteriori los de los controladores implementados. A la hora de realizar la identificación del proceso, se ha obtenido tanto fuera de línea, para calcular el valor de los parámetros del regulador convencional que actúa en la inicialización del sistema, como en línea, para ir calculándose los valores del regulador adaptativo en cada muestra de tiempo.

Bien es cierto que en un principio la identificación fuera de línea no es esencial en los procesos de control adaptativo propuestos, ya que el cálculo de los parámetros de la planta se van realizando en línea, sin embargo marca una clara diferencia en el arranque del sistema cuando el control adaptativo tiene que comenzar a funcionar. Puesto que estos valores, en la inicialización del sistema, son una incógnita que no puede resolverse ya que se necesita un número finito de muestras para poder estimarlos, quizás lo primordial sea intentar conseguir la estabilidad del sistema en esos primeros instantes de tiempo, permitiendo así que el control adaptativo sea capaz de comenzar desde un punto estable.

4.2.1. Identificación de procesos fuera de línea

Aunque MATLAB proporciona algunas herramientas para la identificación, son métodos más enfocados para fuera de línea, es decir, se hace la identificación del sistema trabajando con un rango acotado de datos con los que luego se puede hallar el controlador para unos valores fijos. Algunas de las opciones que MATLAB propone se encuentran en la biblioteca de Simulink *System Identification Toolbox*, mostradas en la ***Figura 4.7***.

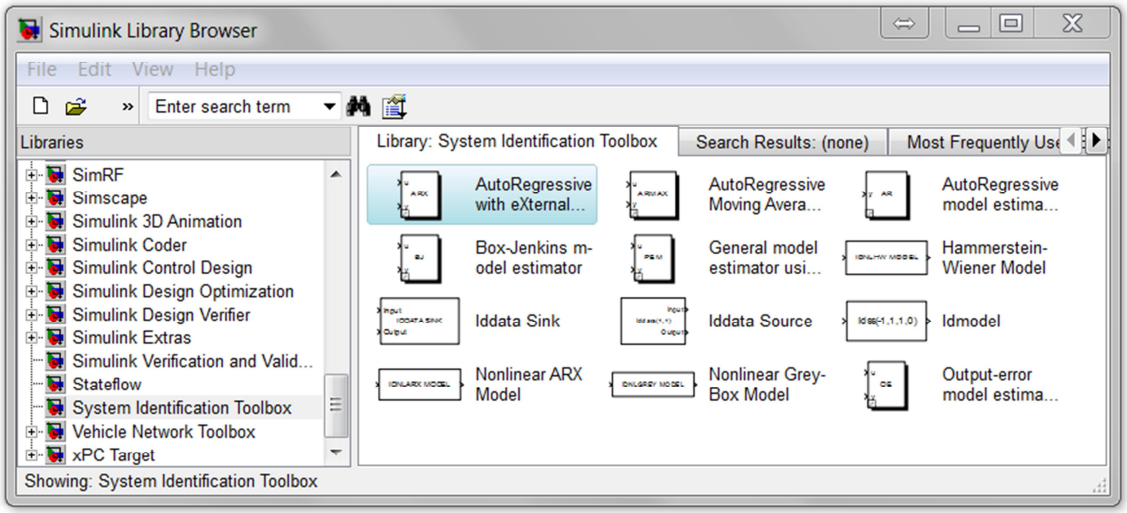

Figura 4.7. Biblioteca System Identification Toolbox

Así estos bloques quedan restringidos únicamente a esta identificación inicial de la planta que permita, dentro del modelo del regulador adaptativo, el cálculo e implementación de un regulador convencional que se ejecute mientras que el sistema se inicia, evitándose de esta manera que se vuelva inestable durante ese número de muestras necesarias para conseguir una primera estimación válida de la planta y que pueda dar paso posteriormente al control adaptativo.

Otra opción para esta primera identificación es el uso de la herramienta ident, mostrada en la ***Figura 4.8***, en la que es necesario disponer de la respuesta del sistema ante una entrada determinada.

En cualquier caso se tienen que obtener previamente una serie de datos del sistema a partir de su comportamiento en frecuencia o su comportamiento ante distintas tensiones de entrada. Esto dará una visión clara de las características de la señal de entrada con la que se debe identificar la planta, además de ciertas no linealidades que aparecen en el caso de un motor de corriente continua, como es el rozamiento de Coulomb, que varía incluso con el sentido de giro, como se puede ver en la ***Figura 4.9***, y que hará que la respuesta del motor no se comporte de forma lineal con la entrada. Una de las principales consecuencias de este fenómeno es que para una serie de valores de entrada de tensión cercanos a cero, el motor no gira. Este efecto no lineal se puede observar en las gráficas del ***Apartado 5.2.3. Resultados del control en velocidad*** cuando se producen cortes con cero.

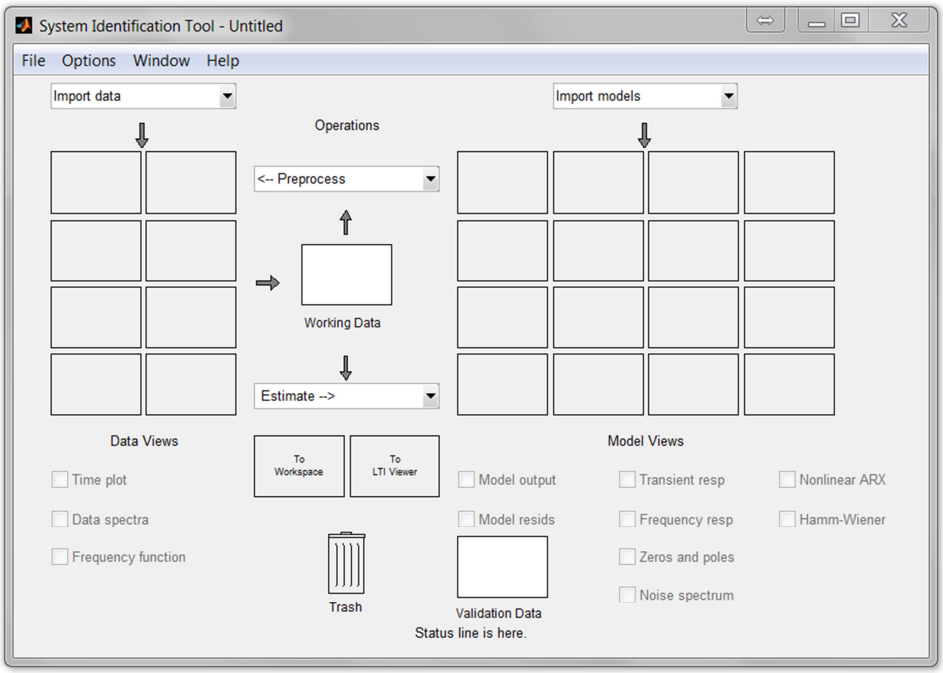

Figura 4.8. Panel Principal de la función Ident

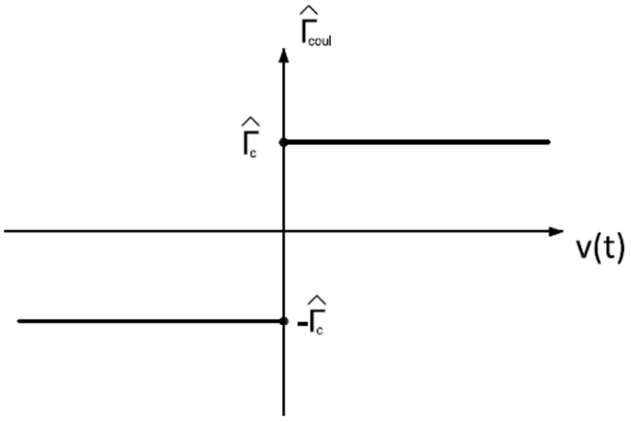

Figura 4.9. Característica de la aproximación al rozamiento de Coulomb

MATLAB ofrece otras opciones, además de existir más métodos de identificación que permiten obtener los parámetros de la planta, en muchos casos, con una gran exactitud, sin embargo estos métodos suelen requerir de un gran número de ensayos experimentales que no suelen estar carentes de complejidad.

Por este hecho y puesto que para los controles adaptativos implementados en el proyecto no es vital el disponer de una identificación exacta de la planta, se ha tomado una solución distinta que, aunque pueda no proporcionar en un primer momento unos resultados tan buenos para la identificación, proporciona un punto medio entre funcionalidad y sencillez, otorgándose para ello una mayor importancia a que el controlador convencional que arranca en los primeros instantes fuerce la estabilidad del sistema, permitiéndole así al controlador adaptativo que comience a trabajar de forma correcta desde un punto estable.

Se pueden obtener los valores de la planta directamente de la función de mínimos cuadrados de los modelos que se han creado, pero puesto que lo que realmente se desea son los parámetros de la planta para poder calcular a posteriori los del regulador, lo que se ha hecho es utilizar estos modelos de controles adaptativos para obtenerlos de forma directa, evitándose un paso. Para ello se lleva al sistema controlado a una zona estable de forma que sea capaz de regularlo a partir de ahí de forma adaptativa. Después se espera un tiempo a que estabilicen los parámetros, con lo que los valores obtenidos son los utilizados para la implementación de los controladores convencionales.

Con esto se obtiene una ventaja importante, y es que si el sistema completo ha conseguido controlarse de forma adaptativa llegado a un punto, significa que un controlador convencional con esos parámetros también es capaz de hacerlo permitiendo la inicialización del sistema.

4.2.2. Identificación de procesos en línea

Simulink no incorpora ningún bloque en sus librerías que permita la identificación del proceso en línea, por lo que es necesario la creación de un bloque que desempeñe este trabajo usando funciones definidas por el usuario y cuya misión es la de poder implementar algún método con el que conseguir alcanzar dicho objetivo. Teniéndose en cuenta los distintos métodos vistos en el ***Apartado 2.1. Identificación de procesos***, se ha decidido la implementación del método de mínimos cuadrados recursivo con factor de olvido, que permite disponer también del método de mínimos cuadrados recursivo haciendo 1 el factor de olvido. Además de tenerse claro el método a implementar se debe saber que la identificación se realiza para obtener funciones de transferencia que tengan la siguiente forma:

$$G(z) = \frac{b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \qquad \textit{(4.1)}$$

Si fuese necesario, como lo es para el caso del regulador PID, se puede hacer 0 alguno de los términos de la función de transferencia b_1, b_2, a_1 o a_2 obteniéndose una función de transferencia con una dinámica equivalente.

Sabiéndose la forma de la función de transferencia que se quiere encontrar y el método que se ha usado para la identificación, la implementación de este se puede explicar en los dos pasos siguientes:

1. ***Generación del vector de regresión –*** Aunque este paso se puede incluir en el siguiente, haciéndose uso de la posibilidad de guardar estados pasados que brinda la S-Función, de esta forma se simplifican los modelos y permite realizar ciertos cambios sin demasiada dificultad pudiéndose adaptar la función de transferencia al control que queremos implementar. Para ello se usa un generador de estados diseñado por MATLAB como el mostrado en la ***Figura 4.10***, cuya salida es un vector que contiene la entrada, la entrada desplazada una muestra, el valor de la salida cambiada de signo y el valor de la salida cambiada de signo, y desplazada una muestra, es decir, su salida es directamente el vector de regresión que se necesita para hallar los valores de la planta por mínimos cuadrados para la función de transferencia descrita en la ecuación ***(4.1)***.

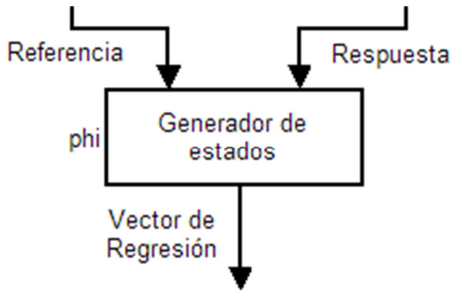

Figura 4.10. Bloque del generador de estados del que se obtendrá el vector de regresión

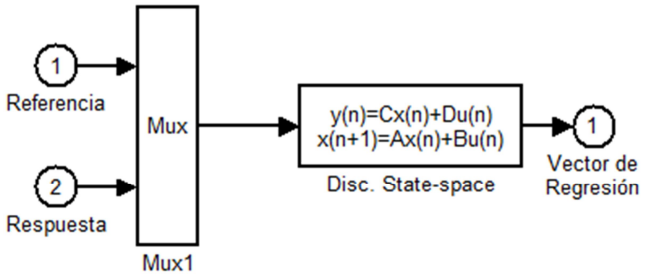

Figura 4.11. Subsistema de que está compuesto el generador de estado

La matriz D del sistema discreto de estados que se puede ver en la ***Figura 4.11***, subsistema contenido en el generador de estados de la ***Figura 4.10***, es una matriz identidad de dimensiones 4x4, en el caso que se estudia, donde cada 1 de la diagonal esta relacionado con uno de los valores del vector de regresión, por lo que si se sustituye algún 1 por 0 ese parámetro del vector de regresión será 0. Esto es especialmente útil ya que cuando se quieren hallar los valores de la planta por mínimos cuadrados, se puede hacer el ajuste evitando alguno de estos términos, puesto que se relacionan a su vez con el vector de regresion, eliminándose si fuera necesario, como lo es en el caso del PID adaptativo con el término b_2, y hallando una función de transferencia equivalente cuyo comportamiento es similar a la que contiene el término.

2. ***Mínimos cuadrados recursivo con factor de olvido –*** Haciéndose uso de una S-Function se implementa el método de mínimos cuadrados recursivos, mostrado en la ***Figura 4.12***, que a través del vector de regresión y de la salida del sistema, obtiene los valores estimados de la función de transferencia de la planta.

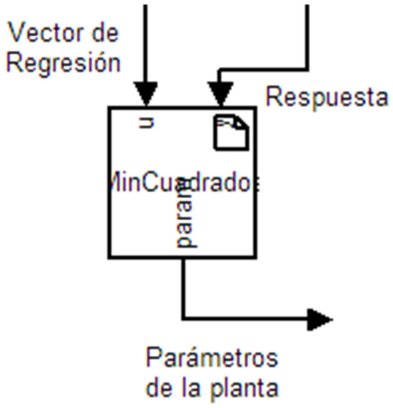

Figura 4.12. Bloque de la S-Function de Mínimos Cuadrados Recursivos

El código de este bloque se puede ver en el ***Anexo C. Código de mínimos cuadrados usando un S-Function Builder.***

De esta forma se tiene que el modelo completo para la identificación de los parámetros de la planta es el mostrado en la ***Figura 4.13***, donde el generador de estados calcula el vector de regresión a partir del cual la función de mínimos cuadrados obtiene estos parámetros estimados.

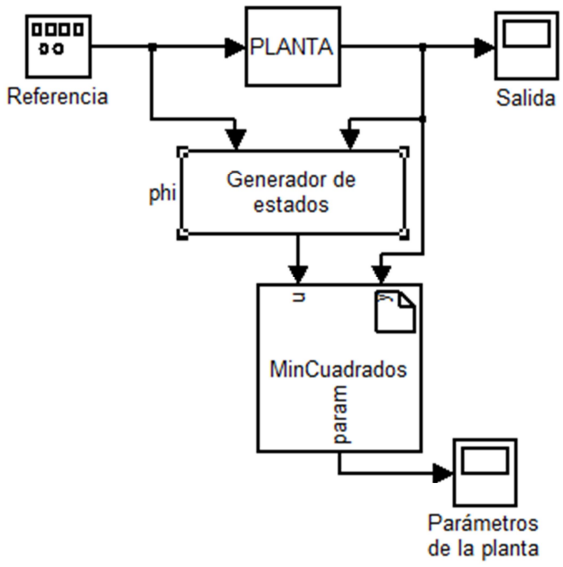

Figura 4.13. Modelo completo para la identificación de un proceso por Mínimos Cuadrados Recursivo

La variable Param mostrada en la ***Figura 4.13***, es un vector de 4 elementos que contiene los parámetros estimados descritos en la función de transferencia de la ecuación ***(4.1)***:

$$param = [b_1, b_{2,} a_1, a_2]' \qquad (4.2)$$

4.3. Control PID adaptativo

Como se describió en el ***Apartado 2.2.1. Regulador PID***, la función de transferencia del regulador PID está definida por la expresión ***(2.45),*** donde los términos de los que depende se hallan a partir de las ecuaciones ***(2.55)***, ***(2.56)***, ***(2.57)***, ***(2.52)*** y ***(2.53)***.

Para que la expresión sea implementable en una S-Function se debe convertir la función ***(2.45)*** de transferencia a ecuaciones en diferencias. Así se obtiene:

$$\begin{aligned} u(k) = u(k-1) + r(k) \cdot (g_0 + g_1 + g_2) \\ - (g_0 \cdot y(k) + g_1 \cdot y(k-1) + g_2 \cdot y(k-2)) \end{aligned} \qquad (4.3)$$

4.3.1. Cálculo del control PID convencional

Los valores de los parámetros del PID convencional que se ejecuta en la inicialización mientras el sistema arranca se obtienen de los que calcula el modelo del PID adaptativo una vez estabilizado, con ello se consigue de forma segura un arranque estable. Algunos de los valores que se pueden utilizar para el controlador PID convencional se muestran en la ***Tabla 5.1*** y la ***Tabla 5.2***.

Hay que hacer uso de unos parámetros del PID que fuercen al sistema a estabilizase. En el caso concreto del motor de corriente continua usando el PID como controlador, los valores hallados para las distintas ganancias k del motor, dan en general buenos resultados.

4.3.2. Implementación del regulador PID adaptativo

El control adaptativo usando un regulador PID se realiza en dos pasos:

1. ***Calculo de los parámetros del PID –*** Se utiliza una función de MATLAB debido a que realizar esta tarea es más sencillo haciendo uso de este bloque que con una S-Funtion y todas las funciones necesarias pueden ser utilizadas en *External Mode*

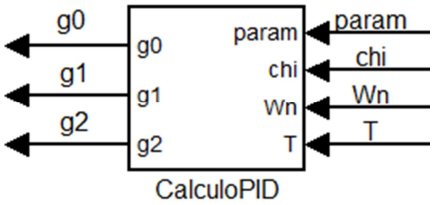

Figura 4.14. Bloque de la función encargada de calcular los parámetros del PID

Las entradas al bloque que realiza el cálculo de los parámetros del PID, mostradas en la ***Figura 4.14***, son:

- ***param –*** parámetros estimados de la planta.
- ***chi –*** factor de amortiguamiento.
- ***Wn –*** frecuencia natural.
- ***T –*** periodo de muestreo.

y las salidas son los parámetros de los que depende el regulador.

El código de la función implementada es el siguiente:

```
function [g0,g1,g2] = fcn(param,chi,Wn,T)

// Obtención de los parámetros estimados del proceso.
b0 = param(1);
b1 = param(2);
a1 = param(3);
a2 = param(4);

// Cálculo de las constantes de tiempo.
t1= -2*exp(-chi*Wn*T*cos(T*Wn*(1-chi^2)^(1/2)));
t2= exp(-2*chi*Wn*T);

// Cálculo de los parámetros del regulador.
g0 = (t1+(1-a1))/b0;
g1 = (t2+(a1-a2))/b0;
g2 = a2/b0;
```

2. ***PID Adaptativo –*** realiza la función del PID adaptativo haciendo uso de los parámetros que se van calculando en cada muestra de tiempo. Para ello se utiliza una S-Funtion, al ser imprescindible, pues según se puede ver en la expresión ***(4.3)*** se necesitan, para calcular la salida en el instante actual, tanto valores pasados como actuales del sistema.

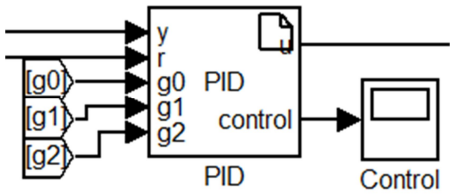

Figura 4.15. Bloque del controlador PID Adaptativo

El control que se muestra en el bloque de la ***Figura 4.15*** indica si está trabajando el regulador convencional o el adaptativo (0 para convencional, 1 para adaptativo).

La configuración elegida para este bloque es:

Número de estados discretos	6
Valores iniciales de los ED	0,0,0,0,0,0
Modo de muestra	Discreto
Valor del tiempo de muestra	0.001

Tabla 4.1. Configuración del S-Function Builder para el bloque del controlador PID adaptativo

Aunque pueda parecer que el tiempo de muestreo es muy pequeño si se tuviese en cuenta la dinámica del sistema, se ha elegido así debido a que la S-Function en el cálculo de los estados discretos no usa realmente el valor actual de las entradas para dar una salida en el mismo instante, si no que usa el valor pasado más reciente. De esta forma cuanto más pequeño sea el tiempo de muestreo, menor es el error cometido al realizar esta operación.

El código de la salida es:

```
u[0] = xD[0];
control[0]=xD[5];
```

y el código del cálculo de los estado discretos:

```
// En el if se ejecutará n muestras el regulador
//convencional, para dar tiempo al sistema que se inicie,
//pasado este tiempo, se ejecutará el else, dando paso al
//PID adaptativo
if ((xD[4]<300)){

    // Los valores de esta función dependerán del sistema
    //que estemos controlando y se obtendrán de una
    //identificación fuera de línea previa de este.
    xD[0] = xD[1]+(4.4784e-04)*r[0]-
            224.4254*y[0]+447.7327*xD[2]-223.3068*xD[3];

    xD[5] = 0; // Control
    xD[4] = xD[4]+1;
 }
```

```
else{
    xD[0] = xD[1]+(g0[0]+g1[0]+g2[0])*r[0]-g0[0]*y[0]-
            g1[0]*xD[2]-g2[0]*xD[3];
    xD[5] = 1; // Control
}

// Se asegura que ha habido alguna señal de entrada, ya que
//si no los parámetros estimados del sistema son 0. No hay
//entrada, no hay salida.
if((r[0]==0)&&(xD[5]==0)){
    xD[4] = 0;
}

// Paso de valores de estado actuales a estados anteriores
xD[1] = xD[0];
xD[3] = xD[2];
xD[2] = y[0];
```

El modelo completo de control con un PID adaptativo interactuando con la tarjeta de adquisición de datos se puede ver en la ***Figura 4.16***. Este modelo se divide en cuatro partes:

1. ***Regulador PID –*** Bloque *"PID"*. S-Function que realiza la tarea de regulador PID adaptativo en función de los parámetros estimados de la planta.
2. ***I / O Analógicas –*** Bloques *"Analog Output"* y *"Analog input"*. Entrada y salida analógicas desde y hacia el motor, permitiendo su control y medida. La función de MATLAB previa a la salida analógica simplemente tendrá el cometido de adaptar la señal de ±5 Voltios a una señal comprendida entre 0 y 10 Voltios. La configuración de la salida de la tarjeta se mantiene a 5 voltios cuando el modelo no se esté ejecutando para evitar que el motor gire.
3. ***Identificación de la planta –*** Bloques *"Generador de estados"* y *"MinCuadrados"*. Su finalidad es la identificación de la planta estimando sus parámetros según se muestra en el ***Apartado 4.2.2. Identificación de procesos en línea***.
4. ***Cálculo de los valores del PID –*** Bloque *"CalculoPID"*. Calcula los parámetros del regulador PID en cada muestra a partir de los parámetros estimados de la planta.

En la ***Figura 4.16*** se puede ver que en la salida del regulador hay un saturador de señal para evitar que la señal de control que llega a la planta sobrepase los valores de ±5 Voltios. Esto en principio no es necesario ya que la tarjeta de adquisición de datos actúa de saturador de forma indirecta, suministrando una tensión a la maqueta comprendida entre 0 y 10 Voltios, que convertida al sistema de referencia del modelo es de ±5 Voltios.

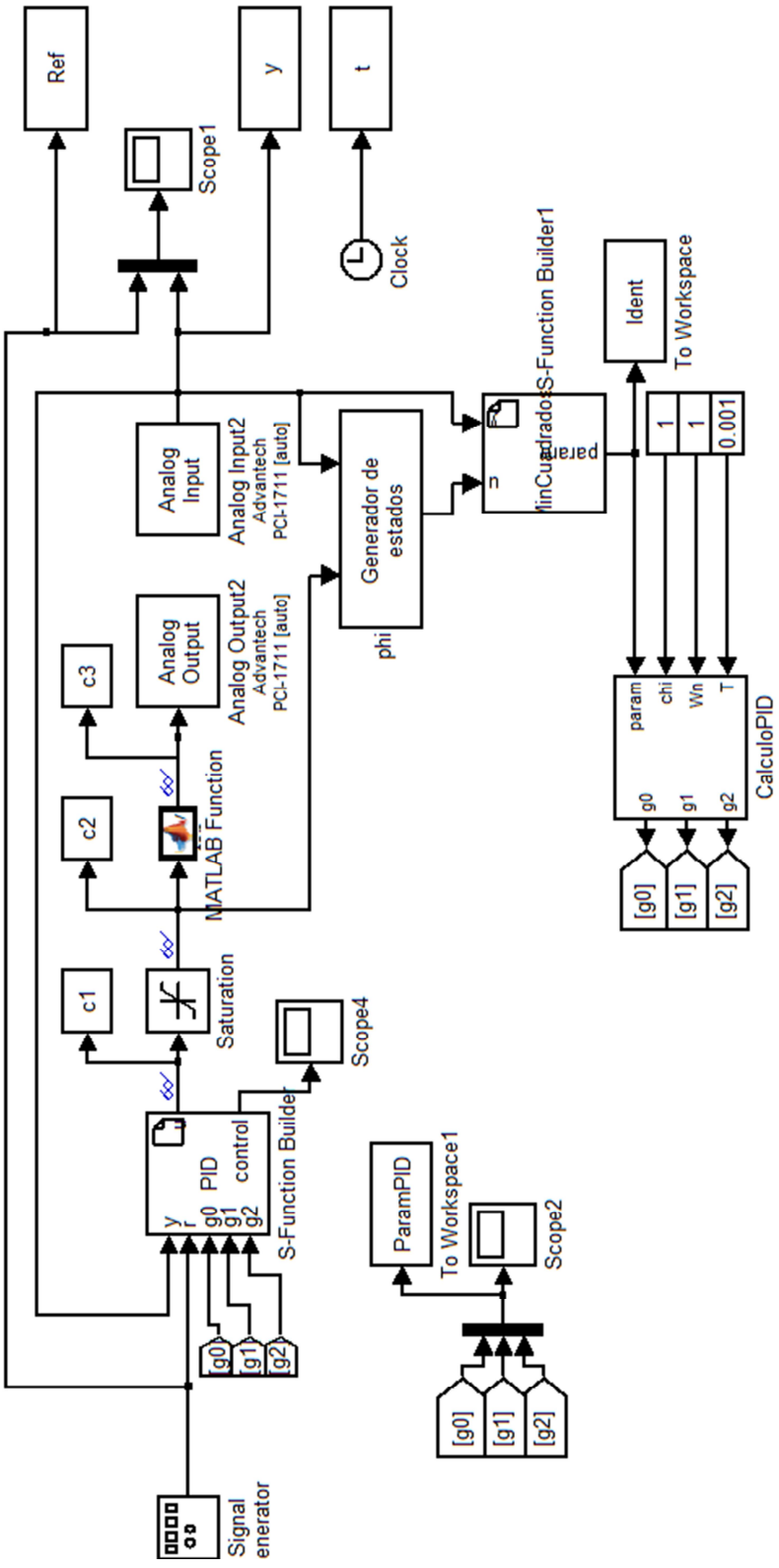

Figura 4.16. Modelo completo de control con PID adaptativo

4.4. Control Dead-Beat adaptativo

La función de transferencia elegida con la que se va a identificar el sistema es de segundo orden con la forma descrita en la expresión *(4.1)*. Esto hace que particularizando la expresión *(2.65)* la función de transferencia del regulador sea la siguiente:

$$G_{db}(z) = \frac{p_0 + p_1 z^{-1} + p_2 z^{-2}}{q_0 + q_1 z^{-1}} \tag{4.4}$$

donde los valores que definen el regulador son:

$$p_i \Longrightarrow \begin{cases} p_0 = \dfrac{r}{\sum b_i} = \dfrac{r}{b_1 + b_2} \\ p_1 = a_1 \cdot p_0 \\ p_2 = a_2 \cdot p_0 \end{cases} \tag{4.5}$$

$$q_i \Longrightarrow \begin{cases} q_0 = r \\ q_1 = -p_0 \cdot b_1 \\ q_2 = -p_0 \cdot b_2 \end{cases} \tag{4.6}$$

Como ocurre con el regulador PID, se debe reescribir en ecuaciones en diferencias para que sea implementable:

$$y(k) = \frac{1}{q_0}(p_0 \cdot u(k) + p_1 \cdot u(k-1) + p_2 \cdot u(k-2) - q_1 \cdot y(k-1)) \tag{4.7}$$

Esto conlleva un problema y es que cada vez que la señal de referencia se hace 0, también se hace 0 el término q_0, con lo que el regulador se vuelve inestable. Para solucionarlo, se ha de retornar a la definición de los parámetros del controlador y obtenerlos en función de un nuevo parámetro k que se define de la siguiente manera:

$$k = \frac{p_0}{q_0} = \frac{1}{b_0 + b_1} \tag{4.8}$$

con lo que se tiene:

$$\frac{p_1}{q_0} = \frac{a_1 \cdot p_0}{q_0} = a_1 \cdot k \tag{4.9}$$

$$\frac{p_2}{q_0} = \frac{a_2 \cdot p_0}{q_0} = a_2 \cdot k \tag{4.10}$$

$$\frac{p_1}{q_0} = \frac{-b_1 \cdot p_0}{q_0} = -b_1 \cdot k \tag{4.11}$$

quedando la expresión final:

$$y(k) = (k \cdot u(k) + a_1 \cdot k \cdot u(k-1) + a_2 \cdot k \cdot u(k-2) + b_1 \cdot k \cdot y(k-1)) \quad \textbf{(4.12)}$$

Se puede observar un hecho notable al reescribir de esta forma los parámetros del regulador y es que si no se tiene un término b_0, estos parámetros no van a depender de la señal de referencia.

Aun así, sigue existiendo la posibilidad de que en la función de transferencia escrita en ecuaciones en diferencias se de una división entre cero y es que b_1 y b_2 sean iguales y de signos opuestos. Una posible solución para aquellos sistemas en los que pueda ocurrir, es tomar una función de transferencia con una forma similar a la que se usa para el controlador PID adaptativo:

$$y(t) = \frac{b_1 z^{-1}}{1 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}} u(t) \quad \textbf{(4.13)}$$

Con lo que al desaparecer el término b_2 este caso ya no se puede dar. Otra opción es la de estimar la función con el término b_0, lo que ocurre es que para ello se debe modificar el sistema de identificación actual.

4.4.1. Cálculo del control Dead-Beat convencional

Para obtenerse los valores iniciales de los parámetros del Dead-Beat convencional se ha seguido el procedimiento que se utilizó para el regulador PID, usar los parámetros que se obtienen del sistema estabilizado del mismo modelo. Algunos de los valores que se pueden utilizar para el controlador Dead-Beat convencional se muestran en la ***Tabla 5.3*** y la ***Tabla 5.4***.

Hay que hacer uso de unos valores del controlador Dead-Beat que hagan que el sistema se estabilice, aunque en el caso concreto de este regulador es más complicado ya que tiende a inestabilizarse con facilidad en los primeros periodos de tiempo, sobretodo actuando en posición.

Para este caso concreto, control en posición, con los parámetros obtenidos con k grandes tiende a estabilizarse casi en cualquier circunstancia, sin embargo el comportamiento al que tiende después, una vez que comienza a funcionar el controlador adaptativo, es peor. Con los parámetros de k medias, el control posterior mejora mucho, pero no es posible iniciar con una k del motor grande pues no es capaz de controlar el sistema.

Como se comenta en el ***Apartado 6.2. Trabajos futuros***, la elección de estos parámetros iniciales bien merece un estudio propio.

4.4.2. Implementación del Dead-Beat adaptativo

El control adaptativo con el controlador Dead-Beat se realiza en dos pasos:

1. ***Calculo de los parámetros del Dead-beat –*** Al igual que en el controlador anterior se utiliza un función de MATLAB para el cálculo de los parámetros del regulador, principalmente por sencillez.

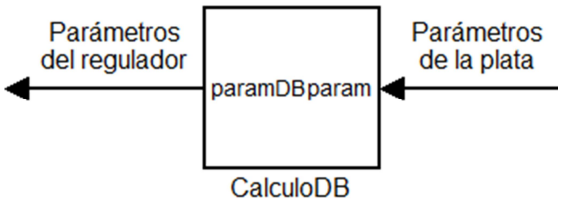

Figura 4.17. Bloque de la función encargada de calcular los parámetros del Dead-Beat

Donde la entrada mostrada en la ***Figura 4.17*** es:

- ***param –*** parámetros estimados de la planta.

y la salida es un vector con los parámetros del controlador Dead-Beat.

El código que contiene la función y realiza los cálculos de los parámetros con los que se ha definido el controlador es el siguiente:

```
function paramDB  = fcn(param)

// Obtención de los parámetros estimados del proceso
b0 = 0;
b1 = param(1);
b2 = param(2);
a1 = param(3);
a2 = param(4);

// Cálculo de los parámetros del regulador.
k = 1/(b0+b1+b2);

paramDB = [0 0 0 0]';
paramDB(1) = b1*k;
paramDB(2) = k;
paramDB(3) = a1*k;
paramDB(4) = a2*k;
```

2. ***Dead-Beat Adaptativo –*** realiza la función del Dead-Beat adaptativo con los parámetros que se van estimando de la planta en cada una de las muestras de tiempo, haciendo uso para ello de la expresión ***(4.12)***. Este bloque se muestra en la ***Figura 4.18***.

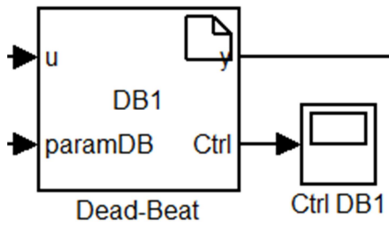

Figura 4.18. Bloque del controlador Dead-Beat Adaptativo

El control indica si está trabajando el regulador convencional o el adaptativo (0 para convencional, 1 para adaptativo), al igual que en el caso anterior.

La configuración elegida para el bloque es:

Número de estados discretos	7
Valores iniciales de los ED	0,0,0,0,0,0,0
Modo de muestra	Discreto
Valor del tiempo de muestra	0.001

Tabla 4.2. Configuración del S-Function Builder para el bloque del controlador Dead-Beat adaptativo

La razón para fijar un tiempo de muestra tan pequeño, es la que ya se mencionó antes, intentar minimizar al máximo el error cometido al realizar las operaciones de cálculo con los estados pasados, en lugar de con los actuales, dentro de la S-Function.

El código de la salida es:

```
y[0] = xD[0];
Ctrl[0] = xD[5];
```

y el código del cálculo de los estado discretos:

```
// En el if se ejecutará n muestras el regulador
//convencional, para dar tiempo al sistema que se inicie,
//pasado este tiempo, se ejecutará el else, dando paso al
//PID adaptativo

if ((xD[6]<300)){


    // Los valores de esta función dependerán del sistema
    //que estemos controlando y se obtendrán de una
    //identificación fuera de línea previa de este.
    xD[0] = xD[1] + 52.8421*u[0] - 83.4726*xD[2] +
           31.4382*xD[3];

    xD[5] = 0; // Control
    xD[6] = xD[6]+1;

}
```

```
else{

    xD[0] = paramDB[0]*xD[1] + paramDB[1]*u[0] +
            paramDB[2]*xD[2] + paramDB[3]*xD[3];
    xD[5] = 1; // Control

}

// Se asegura que ha habido alguna señal de entrada, ya que
//si no los parámetros estimados del sistema son 0. No hay
//entrada, no hay salida.
if((u[0]==0)&&(xD[5]==0)){
    xD[6] = 0;
}

// Paso de valores de estado actuales a estados anteriores
xD[1] = xD[0];
xD[3] = xD[2];
xD[2] = u[0];
```

El modelo completo de control con un Dead-Beat adaptativo interactuando con la tarjeta de adquisición de datos puede verse en la ***Figura 4.19***. Este modelo se divide en cuatro partes:

1. ***Controlador Dead-Beat –*** Bloque *"Dead-Beat"*. S-Function que hace de controlador Dead-Beat adaptativo en función de los parámetros estimados de la planta.
2. ***I / O Analógicas –*** Bloques *"Analog Output"* y *"Analog input"*. Entrada y salida analógicas desde y hacia el motor, permitiendo su control y medida. La función previa a la salida analógica tiene el cometido de adaptar la señal de ±5 Voltios a una señal comprendida entre 0 y 10 Voltios a la que trabaja la tarjeta de adquisición de datos. La configuración de la salida de la tarjeta se mantiene a 5 Voltios cuando el modelo no se está ejecutando para evitar que el motor gire.
3. ***Identificación de la planta –*** Bloques *"Generador de estados"* y *"MinCuadrados"*. Se encargan de la identificación de la planta estimando sus parámetros según se muestra en el ***Apartado 4.2.2. Identificación de procesos en línea***.
4. ***Cálculo de los valores del Dead-Beat –*** Bloque *"CalculoDB"*. Se encarga del cálculo de los parámetros del controlador Dead-Beat en cada muestra a partir de los parámetros estimados de la planta.

En la ***Figura 4.19*** se puede ver nuevamente que a la salida del regulador hay un saturador de señal para evitar que la señal de control que llega a la planta sobrepase los valores de ±5 Voltios, aunque la realidad es que esto no es necesario porque la propia tarjeta realiza la terea de saturador de manera indirecta.

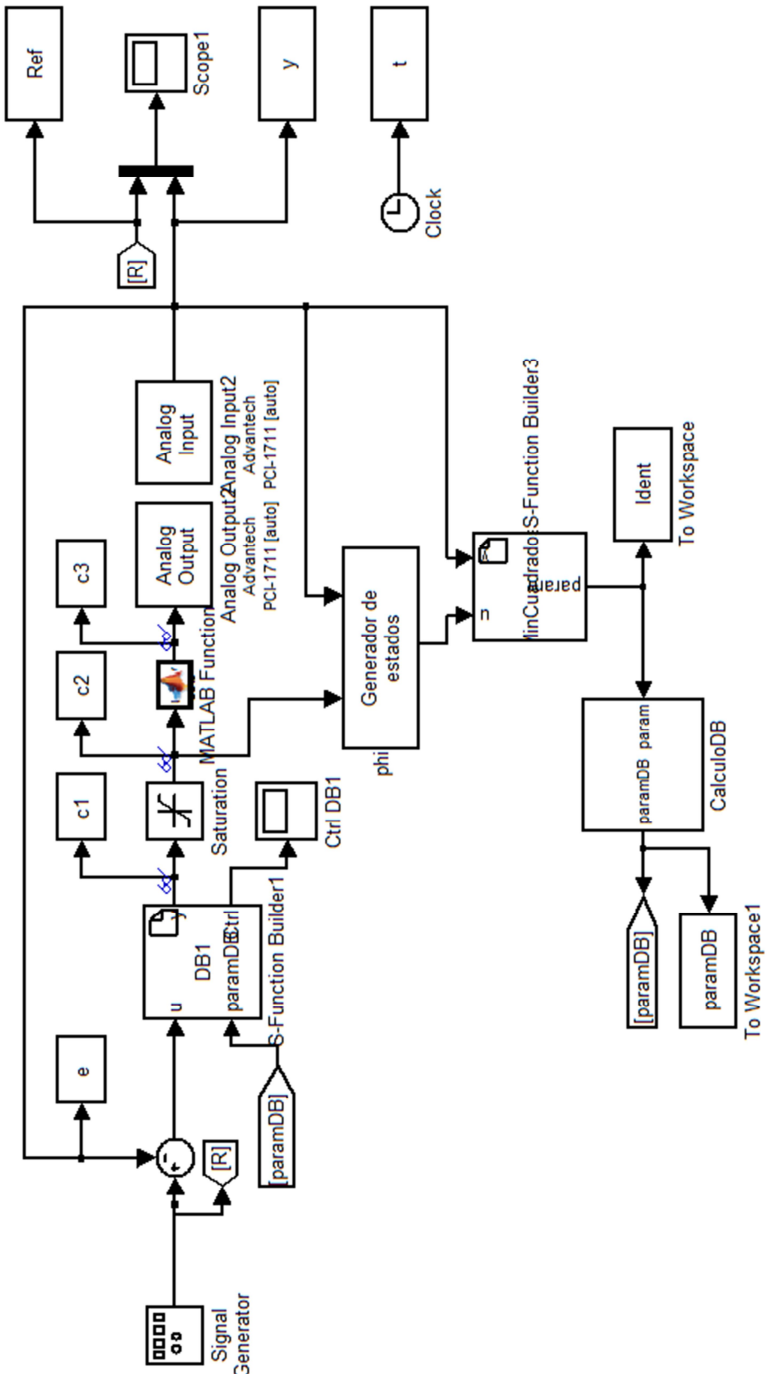

Figura 4.19. Modelo completo de control con Dead-Beat adaptativo

Una vez desarrollados los modelos se procede a realizar los ensayos necesarios para comprobar el funcionamiento de estos. En el siguiente capítulo se observan los ensayos realizados y los resultados obtenidos para cada uno de ellos.

Capítulo 5. Resultados experimentales

Para llevarse a cabo los ensayos se ha usado la plataforma experimental mostrada en la ***Figura 5.1***, compuesta por un PC convencional y por la maqueta que se observa en la ***Figura 5.2***, integrada por el motor de corriente continua eltra con encoder incorporado modelo EH17M1B500712NP1, mostrado en la ***Figura 5.3***, y un circutito acondicionador.

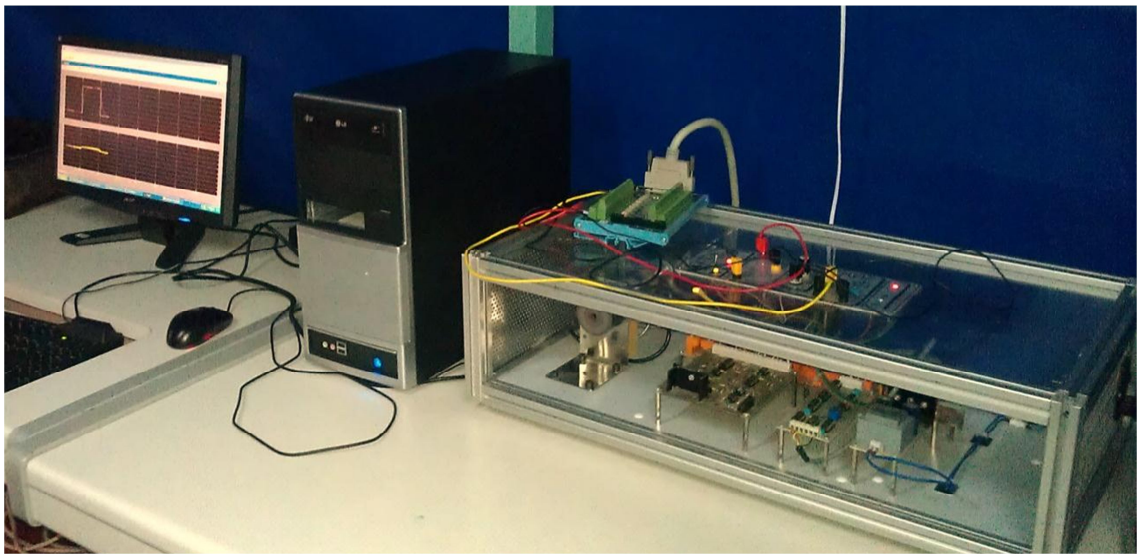

Figura 5.1. Plataforma experimental compuesta de PC, tarjeta de adquisición de datos y maqueta del motor de corriente continua.

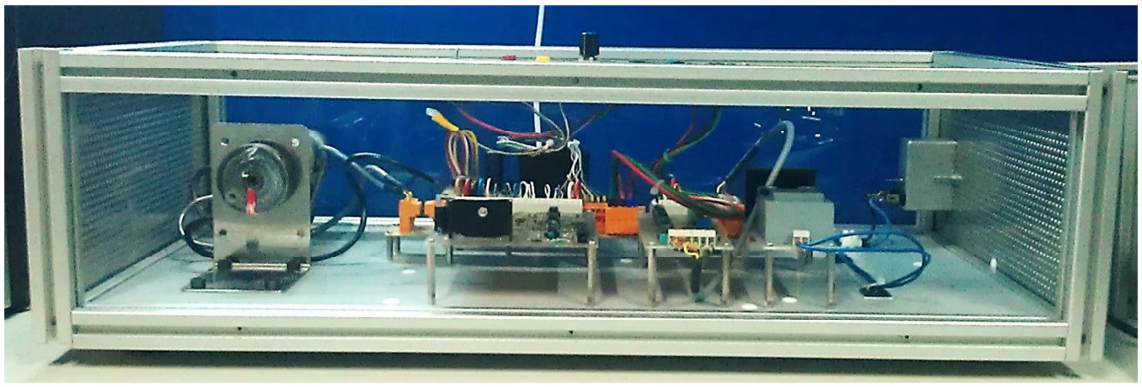

Figura 5.2. Maqueta del motor de corriente continua

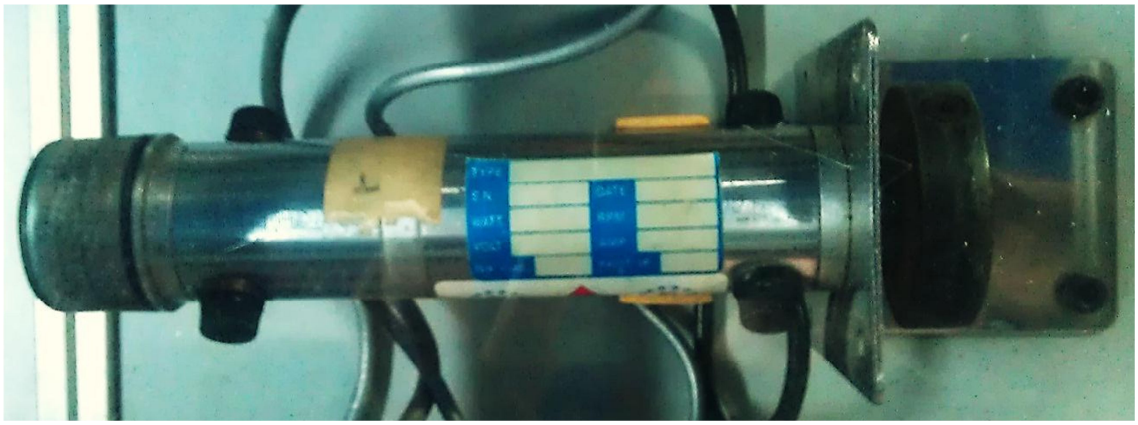

Figura 5.3. Motor eltra con encoder incorporado modelo EH17M1B500712NP1.

Como se puede observar en la ***Figura 5.4*** esta maqueta ofrece la posibilidad de ser configurada para disponer de un control en velocidad o en posición en función del conexionado que se realice, además de permitir regular la constante proporcional del motor gracias a dos diales situados en el panel superior. El primero da la opción de usar tres parámetros fijos de k que se denominarán k_1, k_2 y k_3, mientras que el segundo permite que este ajuste de k varíe entre una constante proporcional mínima k_{min} y una máxima k_{max} de forma continua gracias a un potenciómetro conectado al dial. La relación de estos parámetros viene dada por la expresión:

$$k_{min} < k_1 < k_2 < k_3 < k_{max} \qquad \textbf{(5.1)}$$

Los ensayos experimentales se han realizada variando la ganancia k del motor, de forma que simule cambios en los parámetros de este, con el fin de observar las señales de salida resultante para los diversos valores.

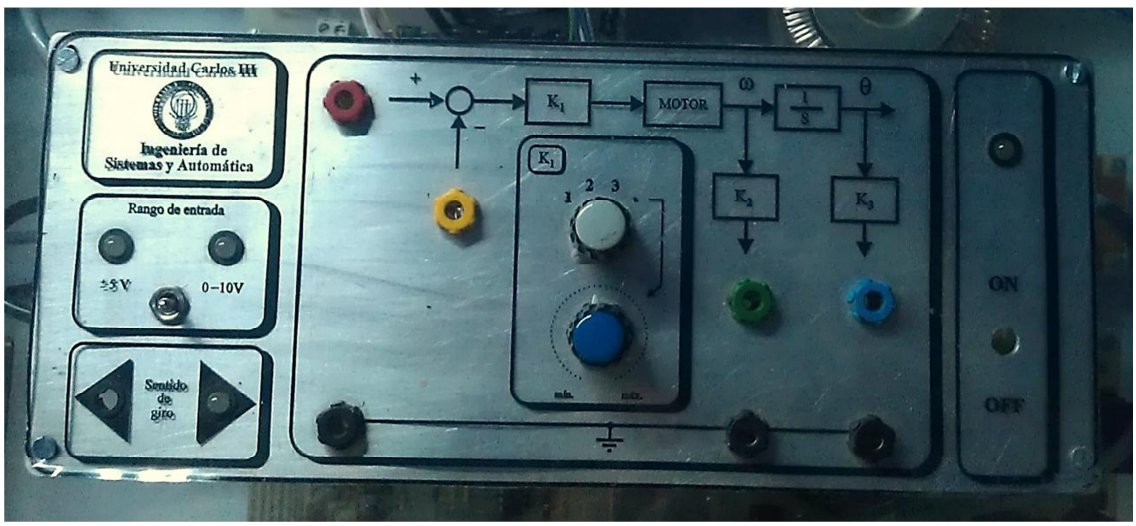

Figura 5.4. Panel superior de conexionado y configuración de la ganancia del motor

La conexión entre el PC y la maqueta, permitiendo el envío y recepción de información, se ha realizado a través de una tarjeta PCI-1711 y el terminal para este modelo PCLD-8710 wiring terminal Board Rev.A1 01-3, mostrados ambos en la ***Figura 5.5***.

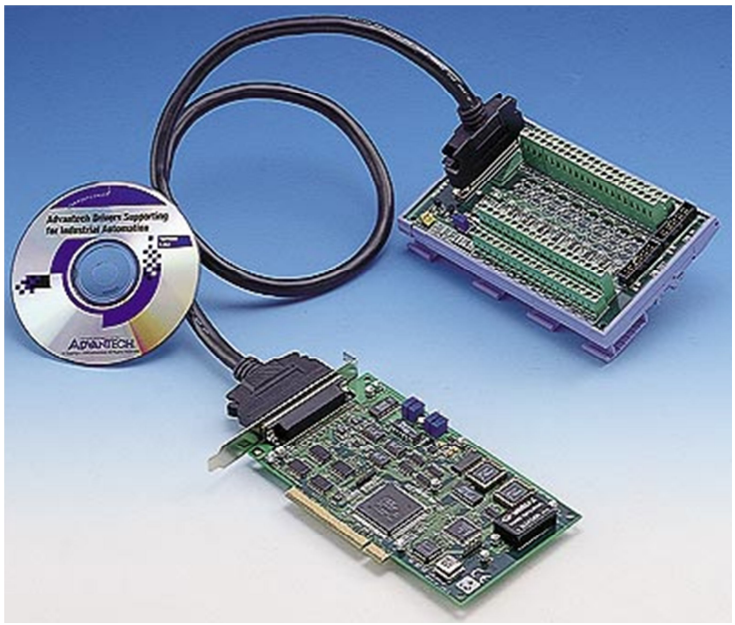

Figura 5.5. Tarjeta PCI-1711 y el terminal PCLD-8710 wiring terminal Board Rev.A1 01-3

5.1. Resultados experimentales con un PID convencional

Este ensayo se ha realizado haciéndose uso del modelo que se muestra en la ***Figura 5.6***, en el que se puede ver un sistema en bucle cerrado compuesto por un regulador PID convencional de parámetros fijos y las I/O analógicas de la maqueta del motor de corriente continua que se va a controlar mediante velocidad, que es la misma que se ha utilizada para todos los ensayos realizados.

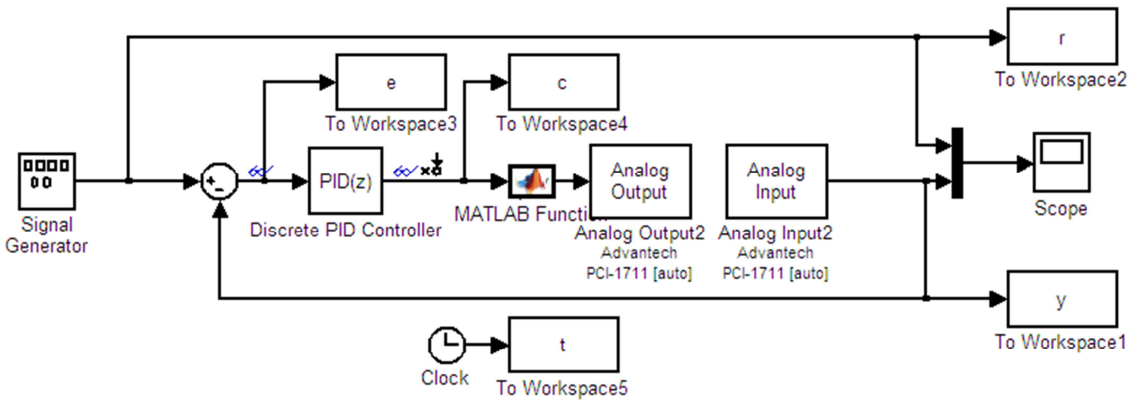

Figura 5.6. Modelo de control en bucle cerrado con PID convencional

La respuesta del sistema se muestra en la ***Figura 5.7*** junto a la señal de referencia. Durante este ensayo se han realizado cambios brusco de la ganancia k del motor, tomando los valores k_1, k_2, k_3 y k_{max}, pudiéndose comprobar que al realizarse estos cambios la señal de salida, respecto a la señal de referencia, se ajusta perfectamente al régimen permanente, ya que el PID persigue hacer el error cero entre la señal de referencia y la de salida, sin embargo otros valores como el tiempo de subida, el valor de pico de la sobreoscilación, el tiempo en alcanzar este valor de pico o el tiempo de establecimiento van cambiando según cambia la k.

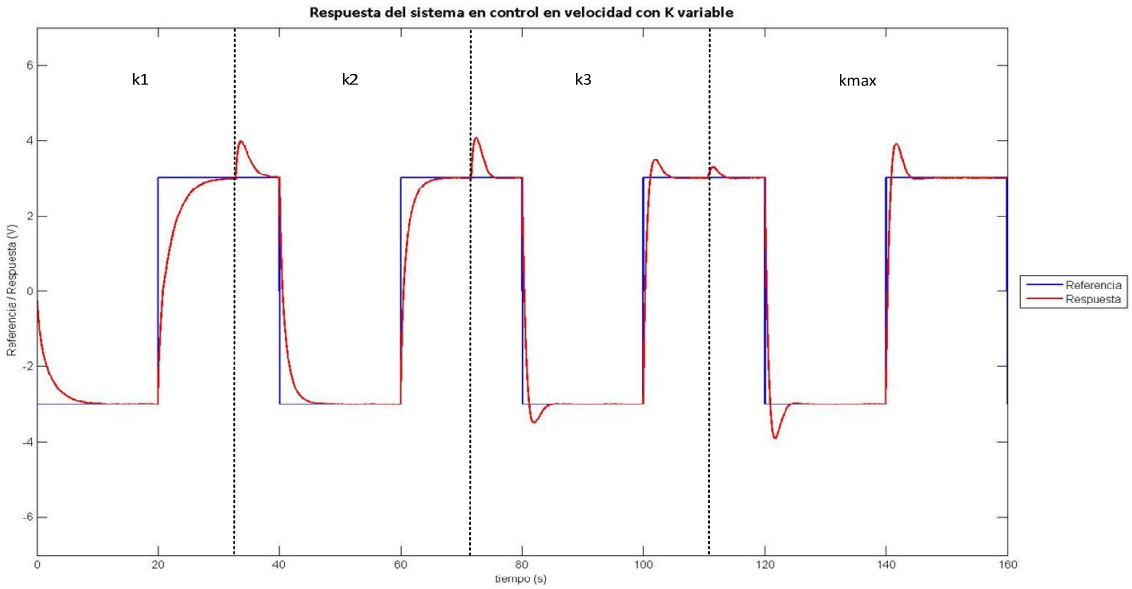

Figura 5.7. Respuesta del control en velocidad con un regulador PID convencional con k variable

En la ***Figura 5.8*** se muestra la señal de control obtenida en el mismo ensayo con el que se ha obtenido la ***Figura 5.7***.

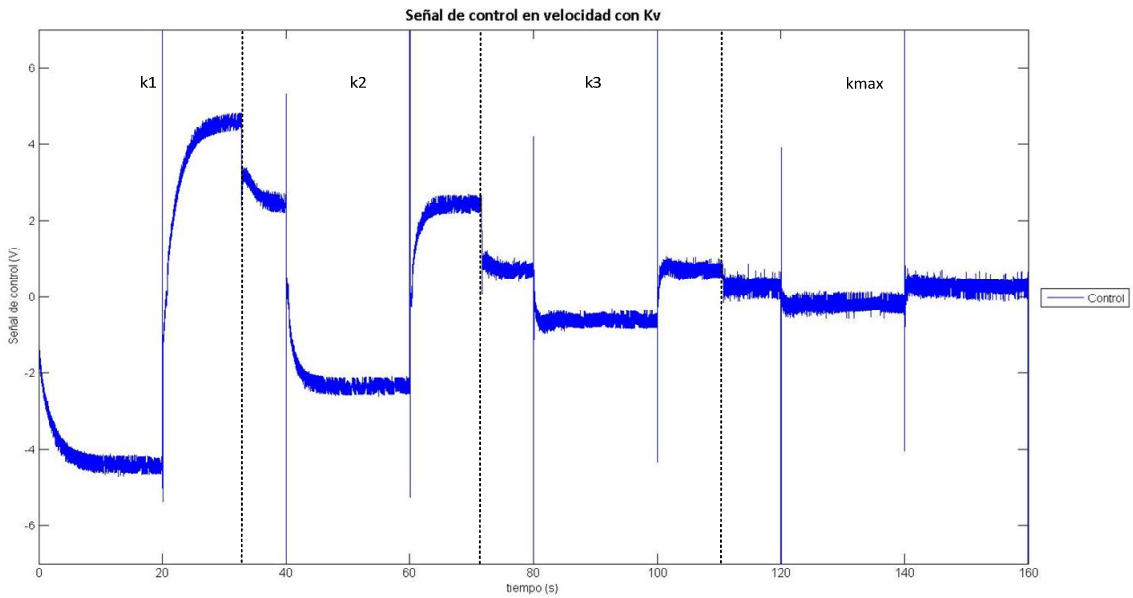

Figura 5.8. Señal de control de un control en velocidad con un regulador PID con Chi y Wn fijas para distintas k.

5.2. Resultados experimentales con un PID adaptativo

Los resultados experimentales de este apartado se han obtenido haciéndose uso de un regulador adaptativo PID que controla la dinámica del conjunto del motor con encoder.

5.2.1. Parámetros iniciales del regulador PID

Para obtenerse los juegos de valores de los parámetros iniciales que componen el controlador PID convencional, integrado dentro del bloque del PID adaptativo, se han realizado tres ensayos en los que se ha variado la constante k del motor a través del primer dial, el cual se ha configurado en cada uno de los ensayos para los valores k_1, k_2 y k_3, tanto para posición, mostrado en la ***Figura 5.9*** como para velocidad, ***Figura 5.10***. Se puede observar que estos parámetros tienden a estabilizarse según va transcurriendo el tiempo, describiendo una señal con forma en dientes de sierra. Estos picos se producen durante el cambio de la señal de referencia, la cual ha sido elegida como una onda cuadrada de período 40s.

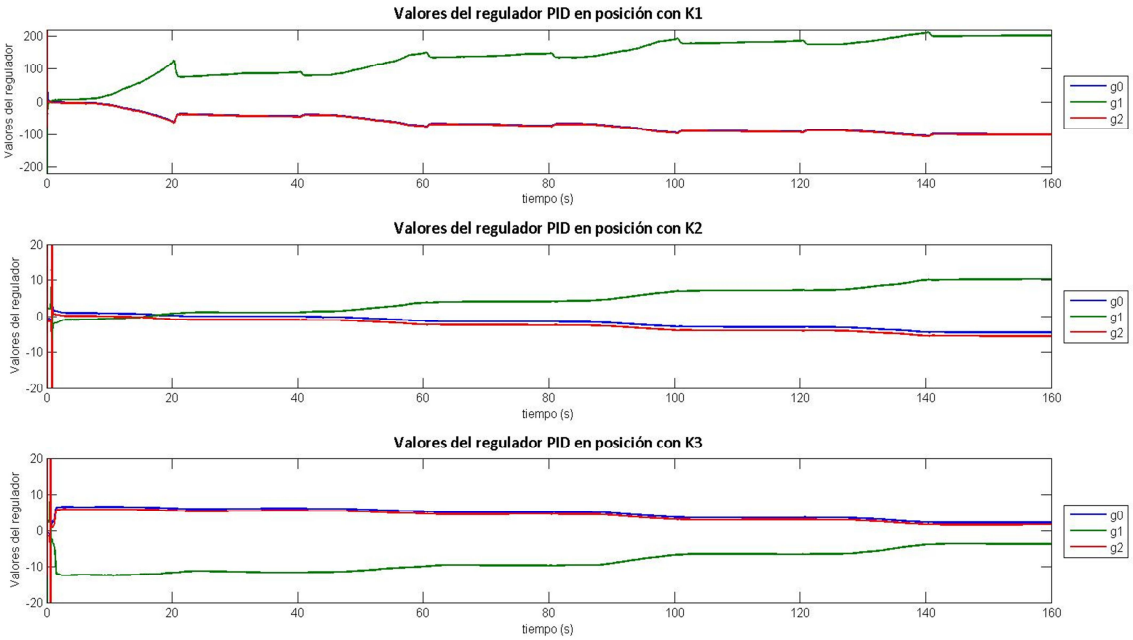

Figura 5.9. Valores del controlador PID actuando en posición para distintas k

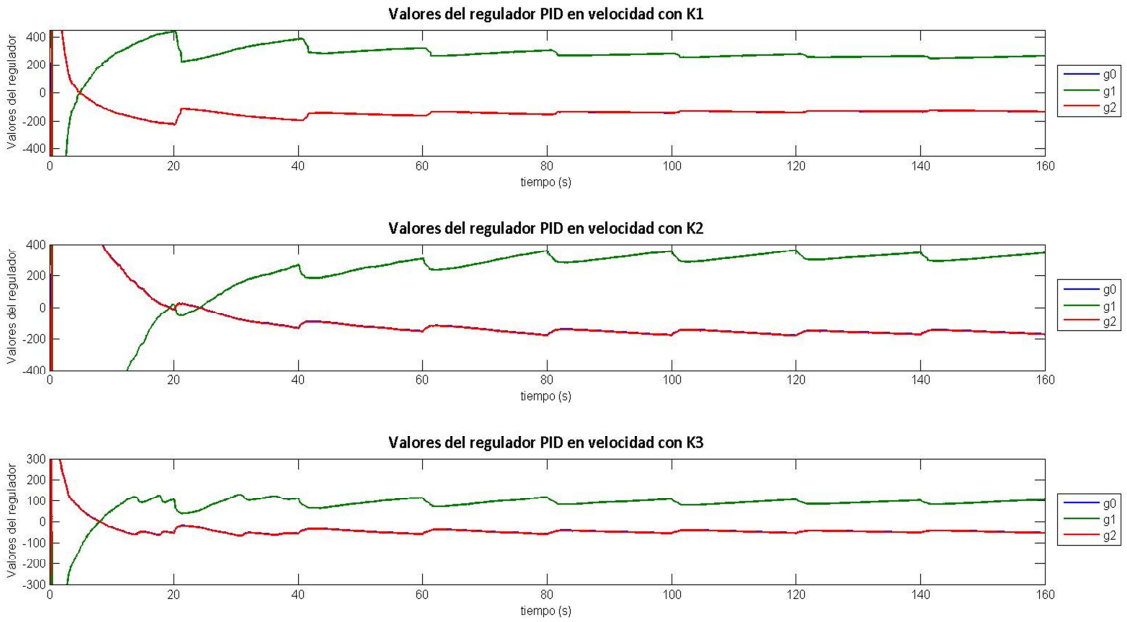

Figura 5.10. Valores del controlador PID actuando en velocidad para distintas k

Los valores estabilizados a los que tiende en control en posición para las distintas ganancias del motor son:

Ganancia	g_0	g_1	g_2
k_1	-99,892	201,622	-101,729
K_2	-4,614	10,254	-5,638
k_3	2,148	-3,730	1,582

Tabla 5.1. Valores para los parámetros del regulador PID, haciendo un control en posición, en función de la ganancia del motor

y los obtenidos en el control en velocidad son:

Ganancia	g_0	g_1	g_2
k_1	-132,762	265,166	-132,403
K_2	-170,867	343,406	-172,538
k_3	-52,670	106,076	-53,405

Tabla 5.2. Valores para los parámetros del regulador PID, haciendo un control en velocidad, en función de la ganancia del motor

5.2.2. Resultados del control en posición

El regulador PID implementado ofrece la posibilidad de realizar cambios de ξ (factor de amortiguamiento) y ω_n (frecuencia natural) que proporcionan distintas respuestas ante la señal de entrada para un mismo parámetro de k. En la ***Figura 5.11*** se puede observar como varía esta respuesta en función de distintos valores de estos parámetros, tomando para ello una k de valor k_2.

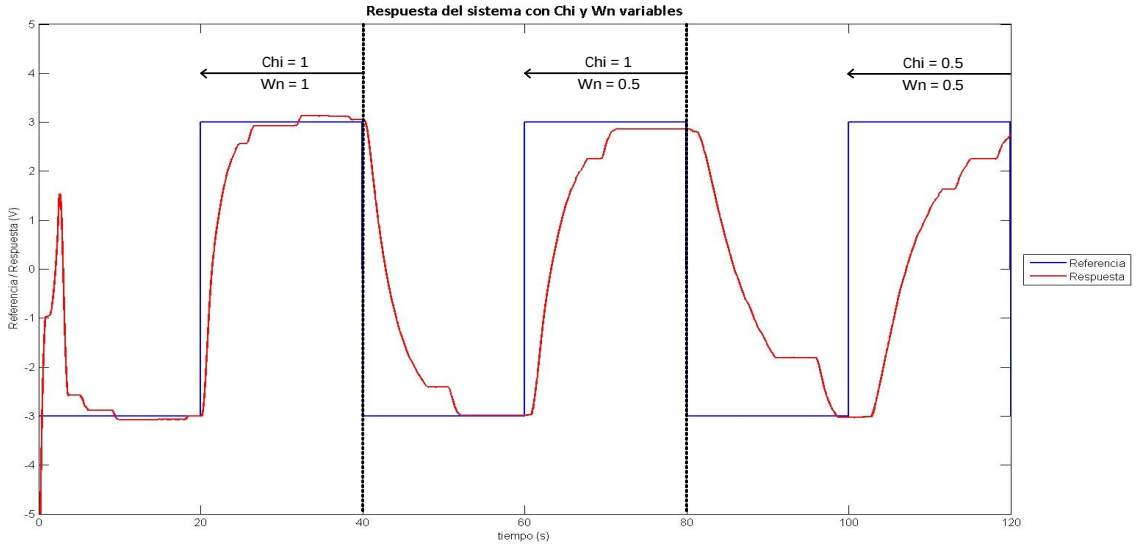

Figura 5.11. Respuesta del control en posición con un regulador PID con Chi y Wn variables

Aunque en principio no se ha realizado ningún control adaptativo en este ensayo, exceptuando quizás el cambio de las primeras muestras de arranque al PID que se va calculando, es interesante poder ver y comprobar que el regulador PID varía su dinámica en función del valor de estos parámetros.

En la ***Figura 5.12*** se puede observar que para una ξ y ω_n fijas, y haciendo cambios en el valor de la ganancia del motor, que se ha configurado para cada uno de los tres ensayos con valor k_1, k_2 y k_3, el comportamiento del sistema es exactamente el mismo. En el caso del motor de corriente continua implica que un cambio de carga, que provocara un cambio en los parámetros de la planta, no afecta a su control, teniendo prácticamente la misma respuesta en los tres casos. Es decir, con distintos parámetros de la planta se consigue la misma dinámica del sistema.

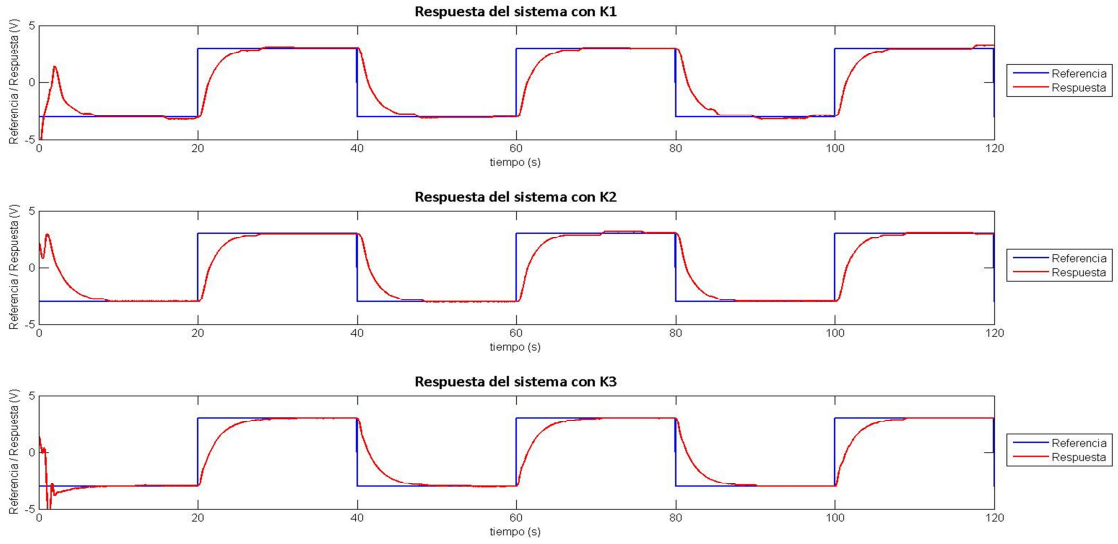

Figura 5.12.Respuesta del control en posición con un regulador PID con Chi y Wn fijas para distintas k

En la ***Figura 5.13*** se muestra la señal de control que se obtuvo al realizar el control en posición, en la que se puede observar que la actuación del controlador es llevada a cabo en los instantes en los que el error cometido crece, estando estos instantes en los cambios de la señal de referencia, siendo el resto del tiempo la señal de control prácticamente cero.

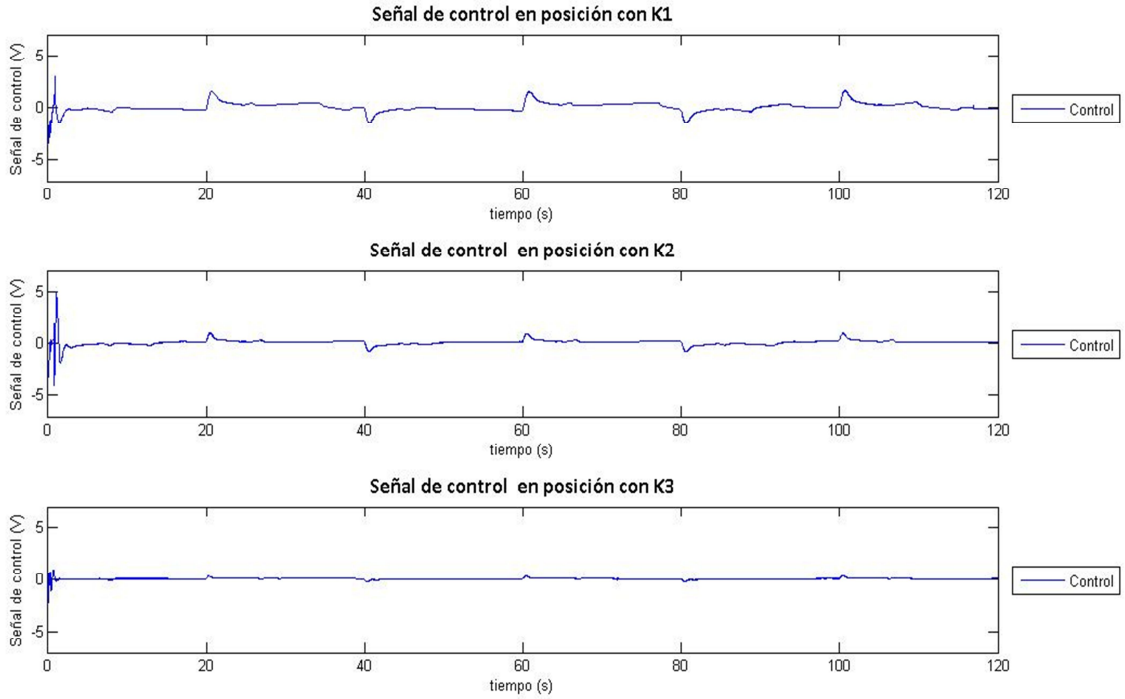

Figura 5.13. Señales de control del control en posición con un regulador PID con Chi y Wn fijas para distintas k

En la ***Figura 5.14*** se muestra el resultado de realizar cambios bruscos de la constante proporcional k del motor durante el ensayo con unos valores una ξ y ω_n fijos en el regulador. En ella se aprecia que el controlador es capaz de adaptarse ajustando la dinámica a los nuevos parámetros. Además se observa que no se producen grandes variaciones en la señal de salida en el instante en el que se produce el cambio de la ganancia. Este hecho se debe a que en el control en posición la señal de control, como puede verse en la ***Figura 5.13***, es prácticamente cero durante todo el tiempo, exceptuando en los instantes que se producen cambios bruscos de la señal de referencia, lo que hace que el control prácticamente no actué.

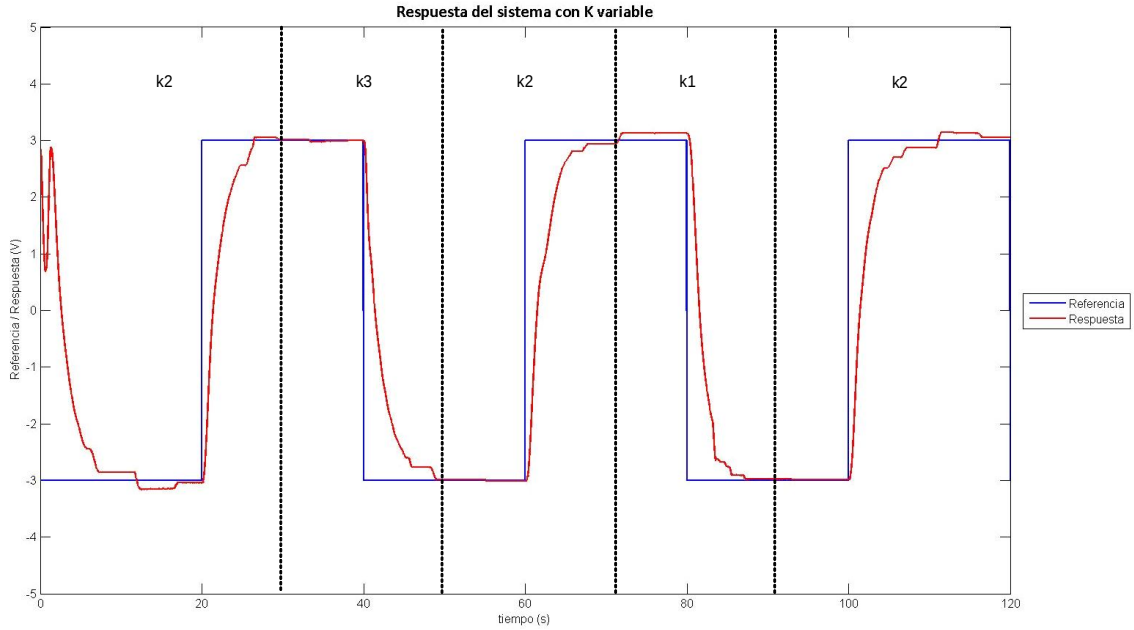

Figura 5.14. Respuesta del control en posición con un regulador PID con Chi y Wn fijas y k variable

En la ***Figura 5.15*** se puede ver un semiperido de la señal de referencia, acompañada de la señal a la salida del sistema y de la señal de control. En esta figura se puede observar que el ajuste de la salida del sistema a la señal de referencia no es exacto, esto provoca que la señal de control comience a crecer lentamente hasta que el motor cambia su posición de forma brusca. Este salto se produce por dos causas, la primera es debida a la baja resolución del encoder utilizado que provoca que el ajuste no sea excesivamente bueno, la segunda se debe a no linealidades como el rozamiento de Coulomb, que impide al motor girar hasta que no se sobrepase la tensión necesaria para vencer este rozamiento.

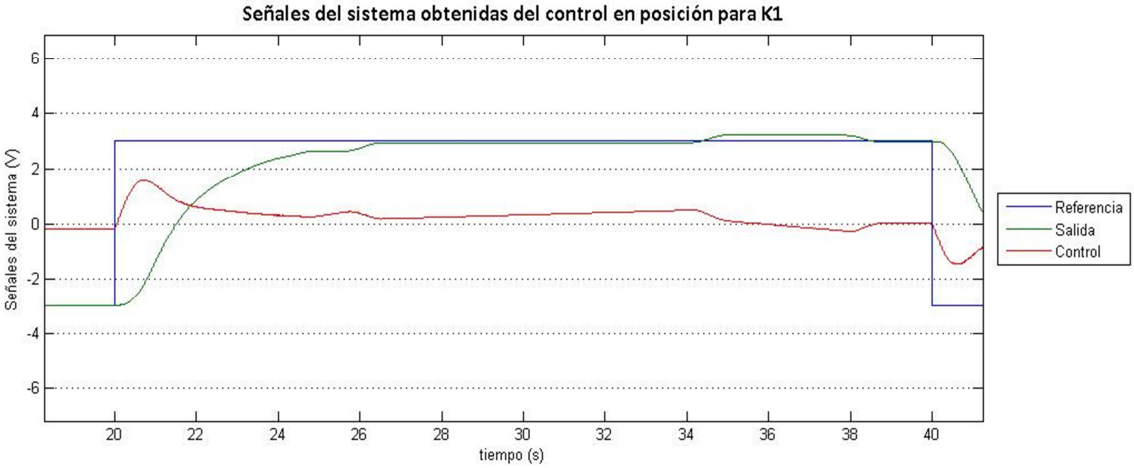

Figura 5.15. Ampliación de un semiperiodo del control en posición con un regulador PID con Chi y Wn fijas y k variable.

5.2.3. Resultados del control en velocidad

En la ***Figura 5.16*** se puede observar como varía la respuesta del sistema con un control en velocidad mediante un regulador PID en función de distintos valores de ξ (factor de amortiguamiento) y ω_n (frecuencia natural), tomando para ello una ganancia del motor k de valor k_2. En esta figura se puede apreciar claramente que en los cortes por cero de la señal a la salida del sistema se producen no linealidades que provocan que para un rango de tensiones pequeñas el motor no se mueva. Esto en principio, aunque no deseable, es normal y se produce en el motor debido a distintos fenómenos no lineales como el rozamiento de Coulomb. Este fenómeno se acentúa cuando el regulador se ajusta para respuestas más lentas ya que las señales de control son menores, hecho que es menos notable en el controlador Dead-Beat al ser de respuesta inmediata, lo que implica que en los cambios de la señal de referencia la señal de control se hace muy grande por lo que el efecto es menor o casi inapreciable.

Se puede comprobar nuevamente en la ***Figura 5.17***, como se vio que ocurre en el control en posición, que en el control en velocidad para los distintos valores de k la dinámica del sistema es la misma.

En la ***Figura 5.18*** se muestran las señales de control obtenida para cada uno de los ensayos con los distintos valores de la ganancia k, observándose una oscilación en dichas señales durante todo el control. Esta oscilación no se debe al control, sino más bien a la baja fiabilidad de la maqueta que se está utilizando. En la ***Figura 5.8***, usando un regulador PID convencional durante todo el ensayo, se aprecia el mismo comportamiento lo que confirma que este fenómeno no es debido al regulador adaptativo.

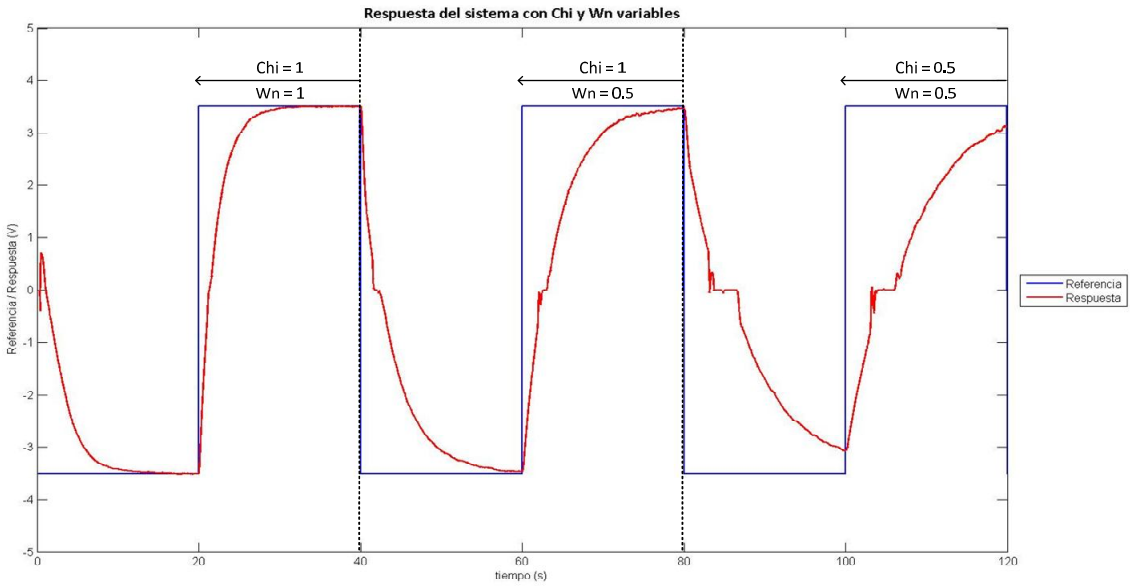

Figura 5.16. Respuesta del control en velocidad con un regulador PID con Chi y Wn variables.

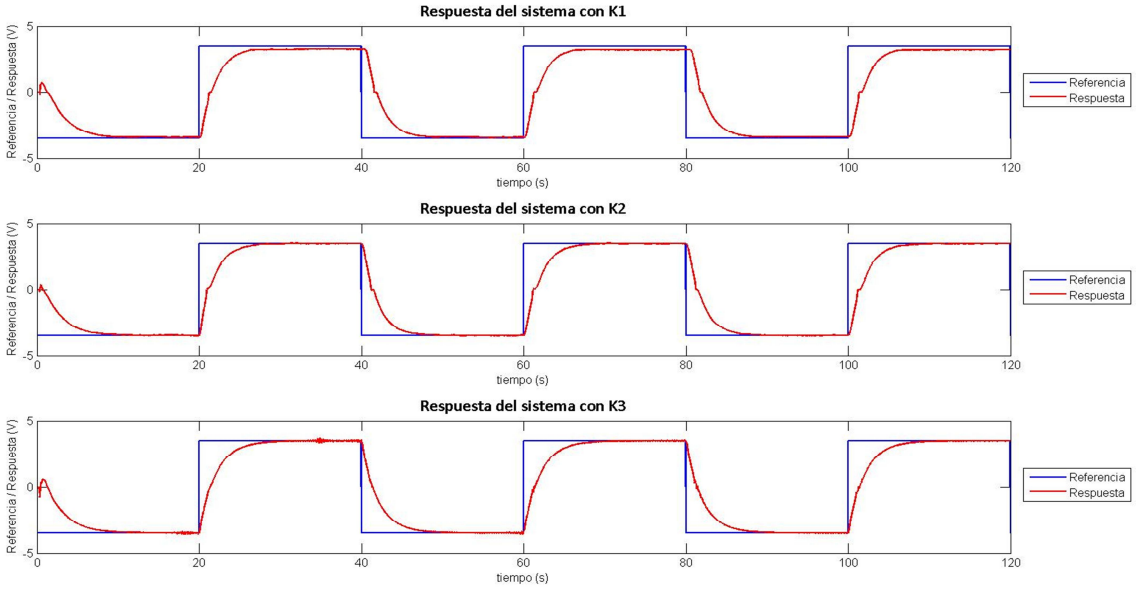

Figura 5.17. Respuesta del control en velocidad con un regulador PID con Chi y Wn fijas para distintas k.

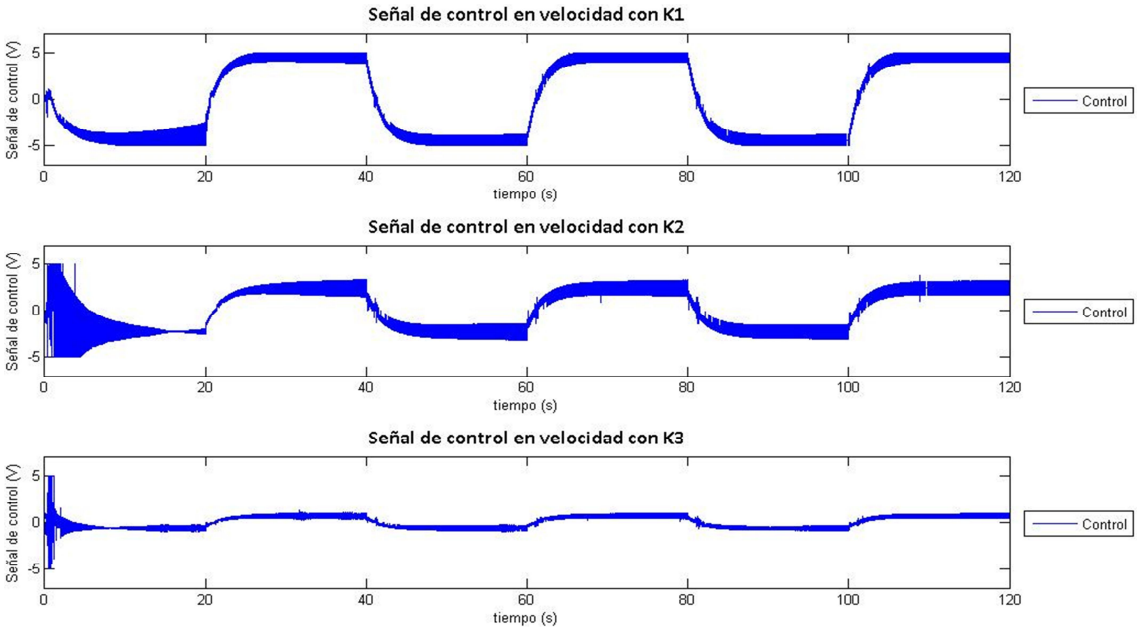

Figura 5.18. Señales de control del control en velocidad con un regulador PID con Chi y Wn fijas para distintas k.

Cuando la planta sufre cambios bruscos en sus parámetros, como se puede ver en la ***Figura 5.19***, el regulador PID adaptativo actuando en control en velocidad vuelve a ajustar sus parámetros para conseguir la dinámica deseada. Para poder realizarse este ensayo se han usado unos valores de ξ y ω_n fijos en el regulador y se ha cambiado la ganancia k del motor en los instantes concretos que se observan en la gráfica.

También puede extraerse de la ***Figura 5.19*** que aunque ante un cambio brusco de k la respuesta se dispare, es capaz de volver a llevar al sistema al estado particular deseado por la señal de referencia. Bien es cierto, que según transcurre el tiempo estos saltos son menores,

producido en parte, por la identificación a través de mínimos cuadrados que tiende a estabilizarse con el tiempo.

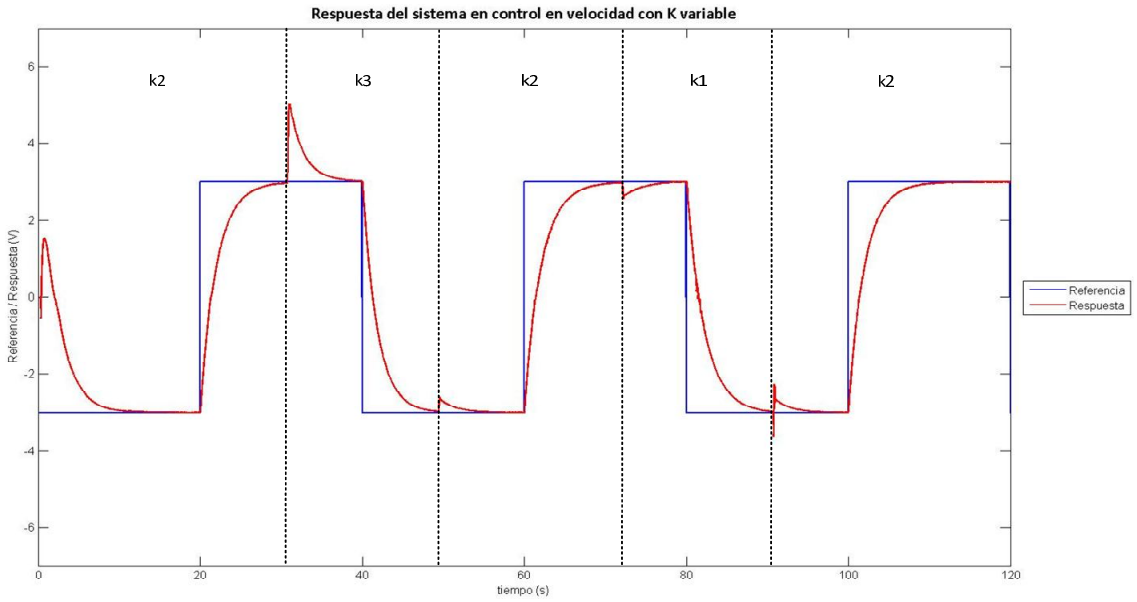

Figura 5.19. Respuesta del control en velocidad con un regulador PID con Chi y Wn fijas y k variable

En la ***Figura 5.20***, se puede ver la señal de control perteneciente a la ***Figura 5.19***, donde se puede observar el cambio en la señal de control en función de las distintas ganancias k del motor usadas durante el ensayo. También se puede comprobar que se ajustan a las señales que se muestran en la ***Figura 5.18*** en cada uno de los distintos tramos, salvo por una diferencia en la amplitud de la oscilación, que es mayor al producirse estos cambios bruscos de k, oscilación que tiende a disminuir según transcurre el tiempo y se estabilizan los parámetros.

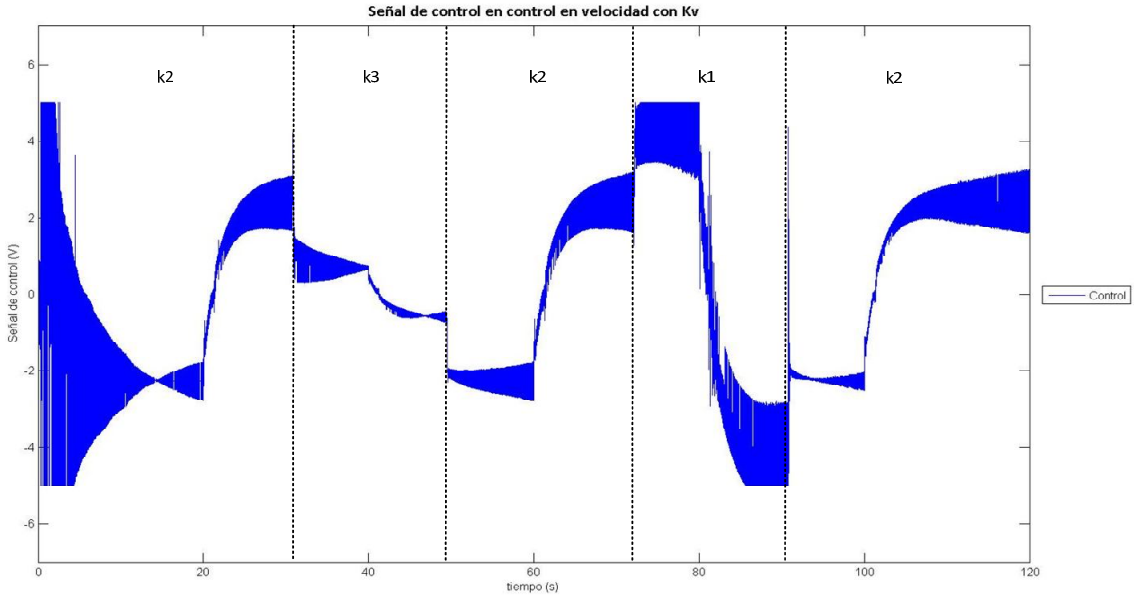

Figura 5.20. Señal de control del control en velocidad con un regulador PID con Chi y Wn fijas y k variable

Se puede comprobar que tanto en posición como en velocidad, observando las gráficas de las repuestas y las señales de control, en los primeros instantes el regulador fuerza la dinámica hasta que es capaz de regular el sistema, punto en el que se comienzan a estabilizar las señales. A partir de este momento el control adaptativo comienza a presentar un comportamiento robusto y fiable. Se comprueba también que el comportamiento del sistema en los tres casos es el mismo, además si se dan cambios bruscos de k, el regulador es capaz de volver a llevar al sistema al estado deseado.

5.3. Resultados experimentales con un Dead-Beat adaptativo

Los resultados experimentales de este apartado se han obtenido haciéndose uso de un regulador adaptativo Dead-Beat que controla la dinámica del conjunto del motor con encoder.

5.3.1. Parámetros iniciales del controlador Dead-Beat

Los juegos de valores de los parámetros iniciales que componen el controlador Dead-Beat convencional que actuará en el arranque del sistema, incluido este dentro del bloque del controlador adaptativo, se han obtenido realizándose tres ensayos experimentales en los que se ha variado la constante k del motor a través del primer dial, el cual se ha configurado para valores k_1, k_2 y k_3, tanto para posición, mostrado en la ***Figura 5.21***, como para velocidad, ***Figura 5.22***. Al igual que en el caso del PID las señales obtenidas describen una forma en dientes de sierra debido al cambio de la señal de referencia, una onda cuadrada de período 40s. Sin embargo se puede observar algo que no ocurría en los parámetros del regulador PID mostrados en las ***Figura 5.9***, y es que para el control en posición, para los valores de k_1 y k_2 de la constante proporcional del motor, el coeficiente $a_2 \cdot k$ del controlador tiende a seguir creciendo más que el resto de parámetros. Esto implica que si se hace uso de estos parámetros para el controlador convencional, se estarán aplicando unos valores que al tardar en estabilizarse tanto en el tiempo pueden dar como resultado un comportamiento inicial que no fuera estable. Sin embargo el uso de los valores estables obtenidos para k_3 dan un comportamiento inicial también estable pero con un mal control adaptativo posterior. Nuevamente se debe llegar a un compromiso entre la dinámica obtenida con el controlador convencional y la obtenida con el adaptativo.

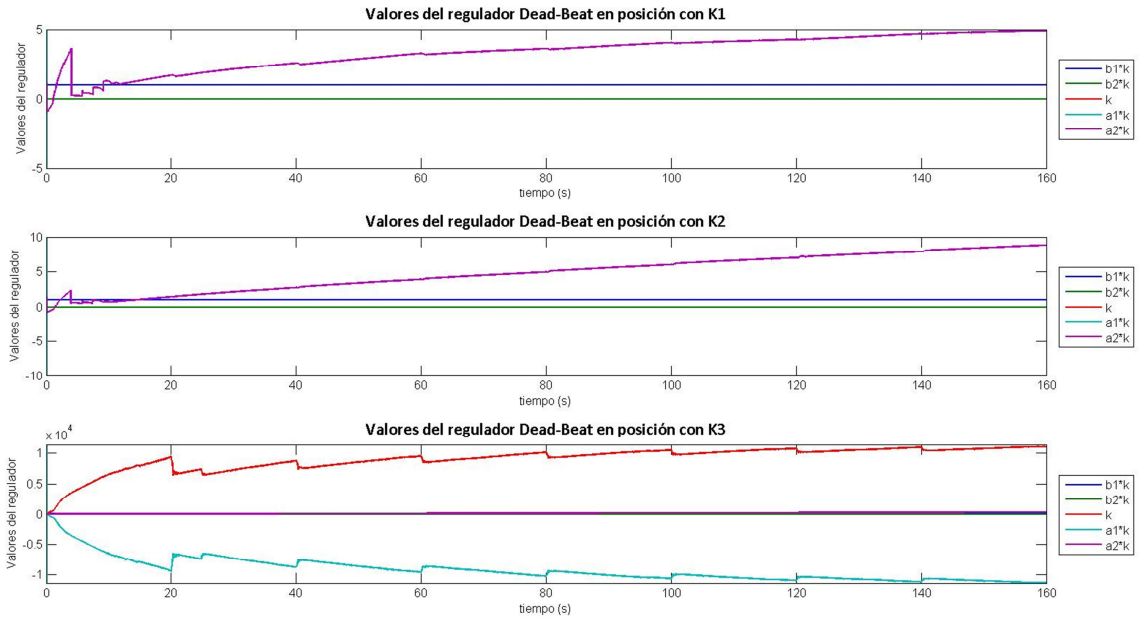

Figura 5.21. Valores del controlador Dead-Beat actuando en posición para distintas k

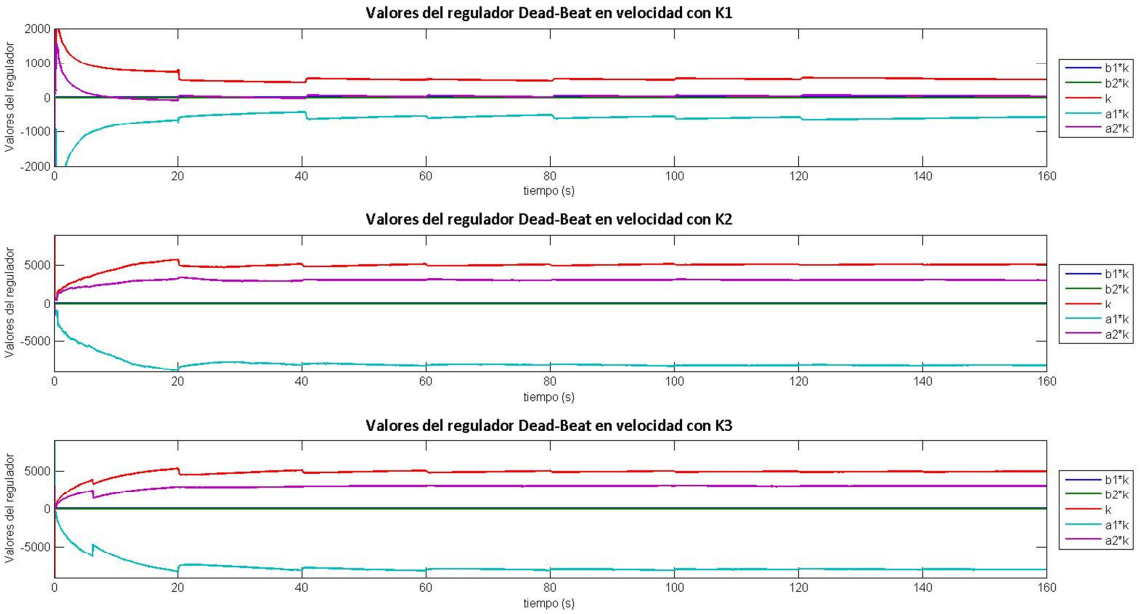

Figura 5.22. Valores del controlador Dead-Beat actuando en velocidad para distintas k

Los valores estabilizados a los que tiende los parámetros del controlador Dead-Beat en control en posición son:

Ganancia	$b_1 \cdot k$	$b_2 \cdot k$	k	$a_1 \cdot k$	$a_2 \cdot k$
k_1	1	0	6508,301	-6508,363	4,913
K_2	1	0	6242,417	-6247,696	8,841
k_3	1	0	11026,113	-11339,627	313,722

Tabla 5.3. Valores para los parámetros del controlador Dead-Beat, haciendo un control en posición, en función de la ganancia del motor

y a los que tiende en control en velocidad son:

Ganancia	$b_1 \cdot k$	$b_2 \cdot k$	k	$a_1 \cdot k$	$a_2 \cdot k$
k_1	1	0	518,420	-557,032	40,117
K_2	1	0	5112,859	-8136,703	3024,645
k_3	1	0	4966,918	-8002,711	3036,445

Tabla 5.4. Valores para los parámetros del controlador Dead-Beat, haciendo un control en velocidad, en función de la ganancia del motor

5.3.2. Resultados del control en posición

En la ***Figura 5.23*** se muestran las respuestas del sistema ante una señal de referencia para distintas ganancias de k, cuyos valores se han fijado como k_1, k_2 y k_3. En los tres casos se consigue la misma dinámica, una respuesta inmediata a cambios de la señal de referencia.

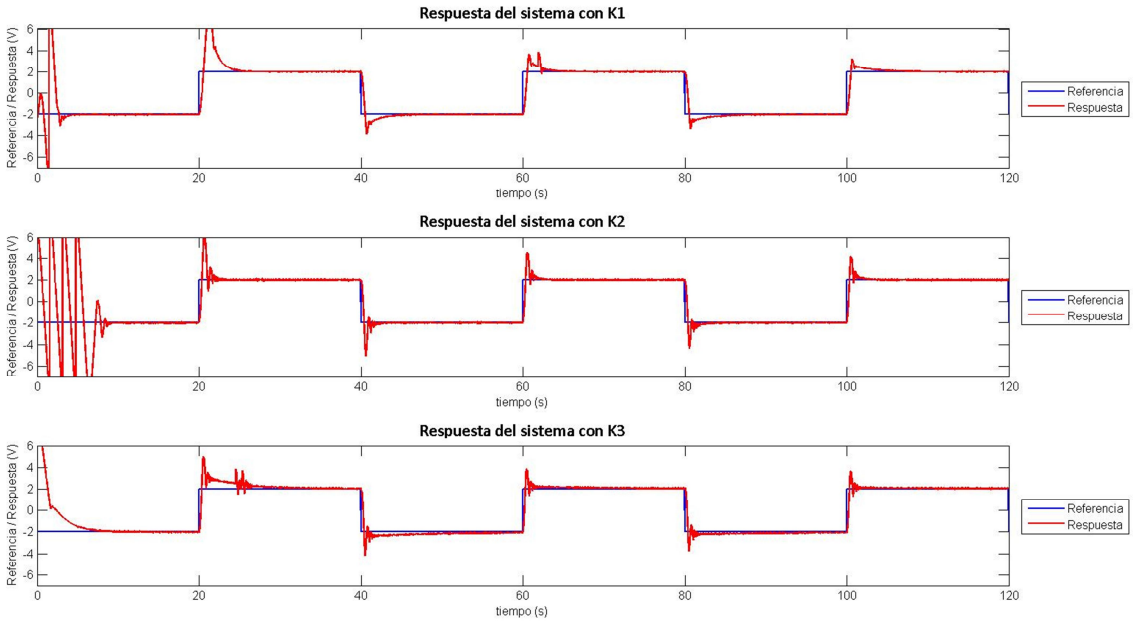

Figura 5.23. Respuesta del control en posición con un controlador Dead-Beat para distintas k

Teniendo además un comportamiento estable a los cambios bruscos de la ganancia del motor según se puede observar en la ***Figura 5.24***.

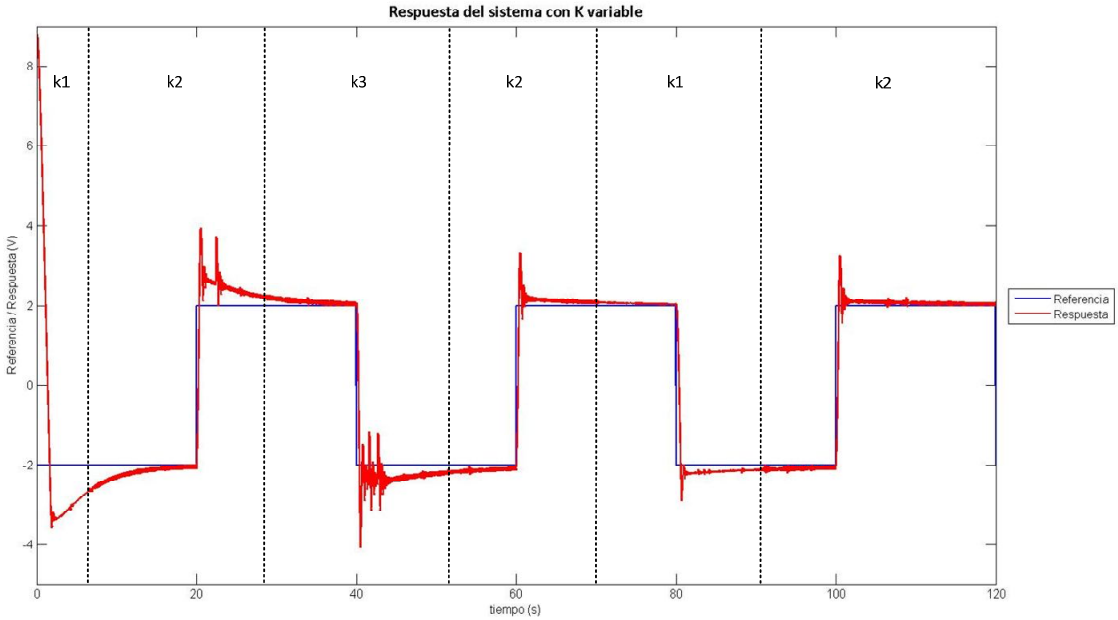

Figura 5.24. Respuesta del control en posición con un controlador Dead-Beat con k variable

Para estos ensayos concretos del Dead-Beat controlando en posición, con el fin de conseguirse que el cambio entre el regulador convencional al adaptativo no inestabilice el sistema, los arranques del sistema se han realizado con un valor de $k = k_1$ de la ganancia del motor en los primeros segundos, pues k distintas provocaban que el regulador adaptativo no fuese capaz de controlar el sistema. Esto se debe a los motivos descritos en el ***Apartado 5.3.1. Parámetros iniciales del controlador Dead-Beat,*** unidos a la propia naturaleza de este controlador.

5.3.3. Resultados del control en velocidad

En la ***Figura 5.25*** se muestran las respuestas del sistema ante una señal de referencia para distintas ganancias de k, cuyos valores se han fijado para k_1, k_2 y k_3. Nuevamente se consigue en los tres casos la misma dinámica, una respuesta inmediata a cambios de la señal de referencia. Se puede apreciar además que las sobreoscilaciones en la salida debido al seguimiento de la señal de referencia se van atenuando según transcurre el tiempo, llegando a ser muy pequeñas. Como ya se comentó en los ensayos para el regulador PID adaptativo, esto tiene en parte su origen en los parámetros estimados en el método de mínimos cuadrados.

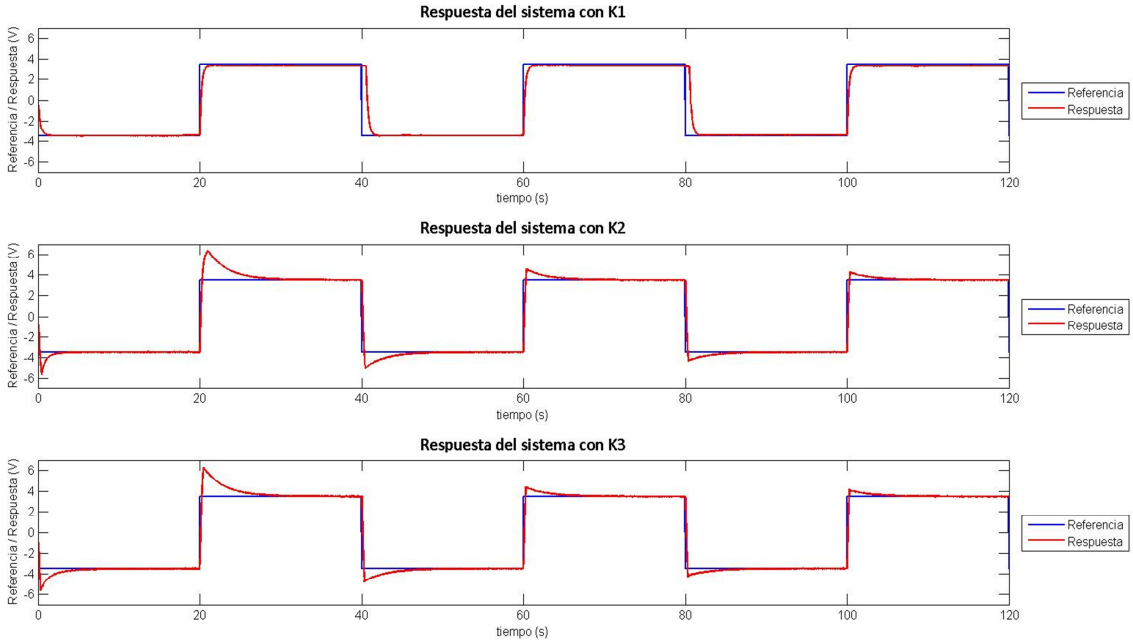

Figura 5.25. Respuesta del control en posición con un controlador Dead-Beat para distintas k

En la ***Figura 5.26*** se muestra el ensayo realizado en el que se lleva a cabo cambios bruscos de la ganancia k, donde se puede ver que el sistema es capaz de adaptar su dinámica a estos nuevos valores de la planta.

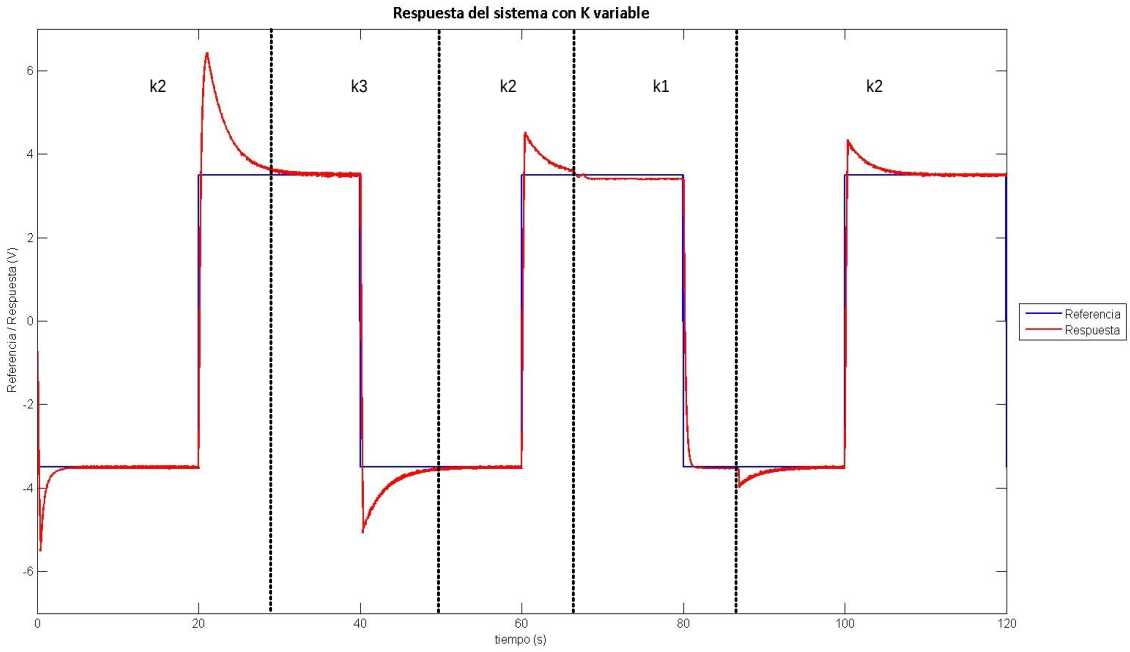

Figura 5.26. Respuesta del control en posición con un controlador Dead-Beat con k variable

Si se observa la ***Figura 5.23*** y la ***Figura 5.25*** se comprueba que aunque la dinámica del sistema se ajuste a una respuesta inmediata, la sobreoscilación producida al seguir la salida del sistema a la señal de referencia varía en mayor o menor medida en función de la k del motor. Este hecho se debe a que este controlador pretende conseguir un tiempo finito de establecimiento sin importar otros parámetros, por lo que este comportamiento está ligado a la propia naturaleza del controlador y reduce el número de usos reales en los que se puede aplicar, y más si la variabilidad de los parámetros de la planta es grande.

Además se produce otro efecto, en general no deseable en aplicaciones reales, del que ya se habló en el ***Apartado 2.2.2. Controlador de tiempo de establecimiento finito (Dead-Beat)*** y es que debido a que se requiere una respuesta Dead-Beat sólo en los instantes de muestreo, pueden producirse oscilaciones que no decrezcan en la respuesta permanente entre dichos instantes, incluso siendo el sistema de control internamente estable. Esto se puede ver en la ***Figura 5.27*** donde se aprecia unos valores de la señal de control para el control en velocidad muy elevados, la señal obtenida en el control en posición es muy similar, si se tiene en cuenta que la señal de referencia que se introduce al sistema está comprendida entre ±3.5 Voltios, pero sin embargo el control es estable como se muestra en la ***Figura 5.23*** y la ***Figura 5.25***. Puesto que la tarjeta de adquisición de datos trabaja para un rango de tensiones de 0 a 10 Voltios, convertido a la referencia de tensiones del modelo un rango ±5 voltios, significa que sobre la señal de control está actuando un saturador de forma indirecta. Si observamos la ***Figura 5.28***, que es una ampliación la señal de control saturada para este rango de valores, se puede ver que en esencia lo que el control Dead-Beat está haciendo es una control a través del ancho de pulso que le envía al motor, lo que provoca que un instante esté mandando +5 voltios y en el instante siguiente -5 voltios, lo que en un entorno real reduciría la vida de la planta que se está controlando.

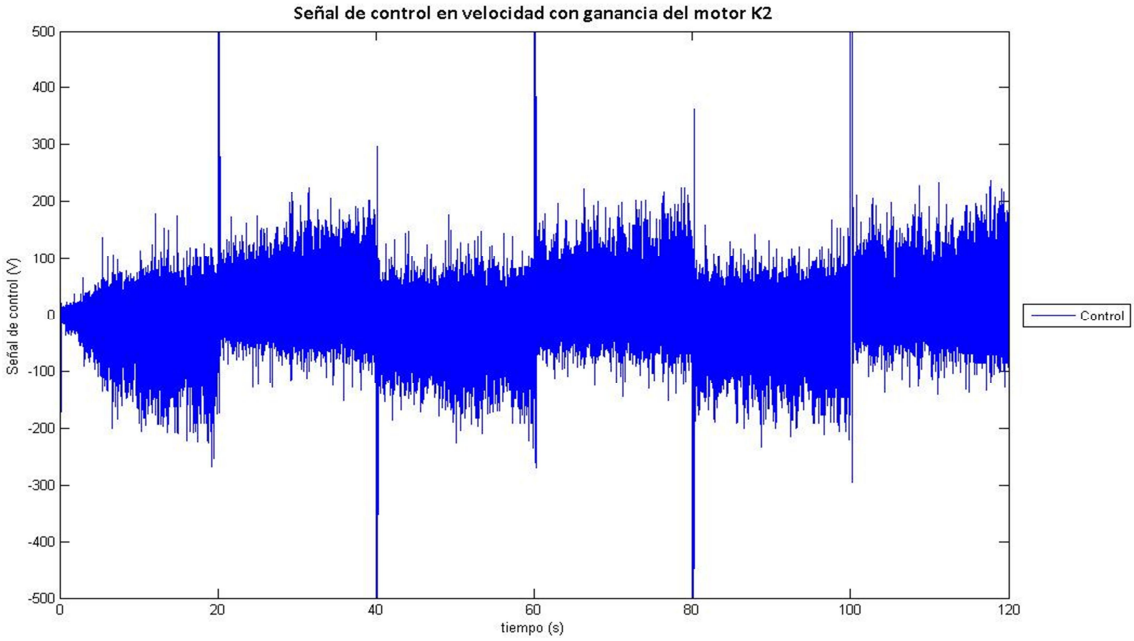

Figura 5.27. Señal de control obtenida del control en velocidad con un regulador Dead-Beat Adaptativo

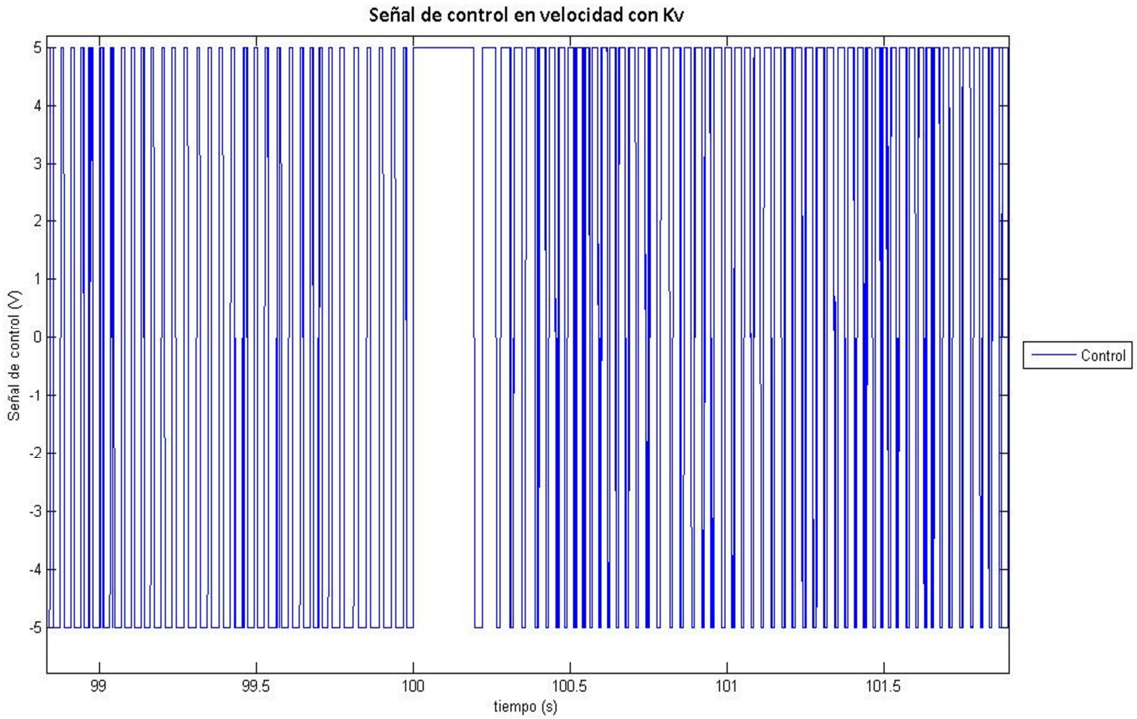

Figura 5.28. Ampliación de la señal de control saturada obtenida del control en velocidad con un regulador Dead-Beat Adaptativo

5.4. Comparativa del control convencional frente al adaptativo

Para la obtención de los resultados de este ensayo se han realizado cambios bruscos de la constante proporcional k del motor, tomando esta ganancia los valores k_1, k_2, k_3 y k_{max}, tanto para el controlador convencional como el adaptativo a fin de poder compararse ambos, produciéndose estos cambios en los mismos instantes de tiempo.

Observándose la ***Figura 5.29***, en la que se encuentran la respuesta a una señal cuadrada de entrada, haciéndose uso para ello tanto del PID convencional mostrado en la gráfica superior como del PID adaptativo en la gráfica inferior, se puede comprobar que en ambos casos el controlador tiende a hacer cero el error en régimen permanente, sin embargo con el convencional la dinámica del sistema tiende a cambiar en función de la ganancia del motor, hecho que no se da en el control adaptativo, el cual es capaz de mantener la misma dinámica durante el régimen transitorio independientemente del valor de dicha ganancia.

La implementación del regulador PID adaptativo con el modelo utilizada tiene una desventaja frente al controlador PID convencional y es que, según se puede ver en la ***Figura 5.30***, la señal de control empeora. Esto es un error subsanable en gran medida, cuya solución empieza por optimizar el método de identificación usando alternativas más fiables respecto al propuesto en dicho proyecto. En el ***Apartado 6.2.2. Identificación del sistema en línea*** se proponen algunas alternativas que pueden ayudar en esta mejora.

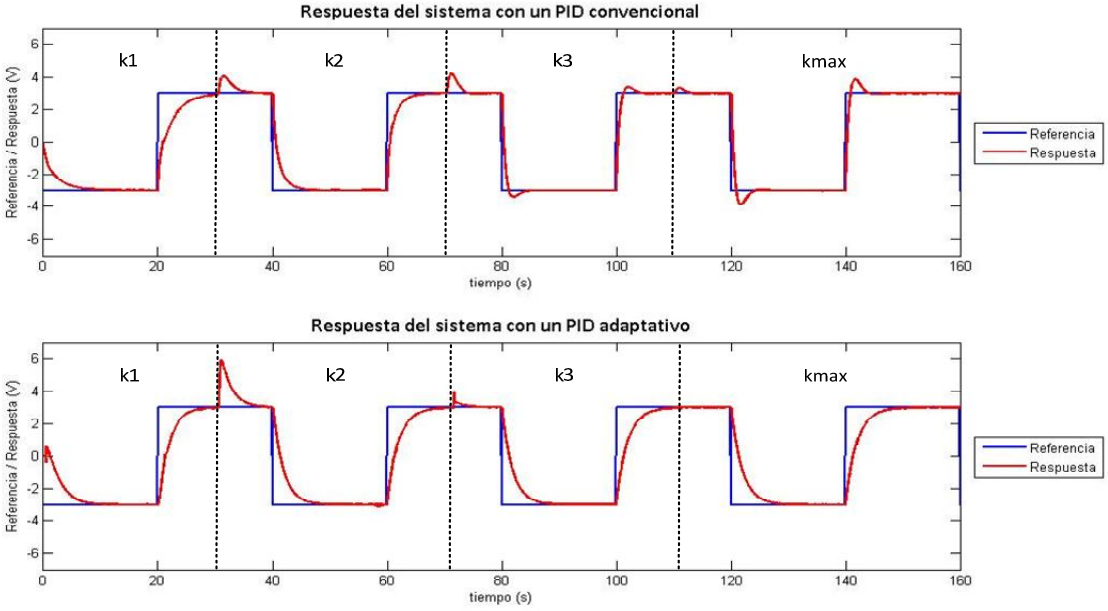

Figura 5.29. Respuesta del control en velocidad con un regulador PID convencional y con un regulador PID adaptativo con k variable

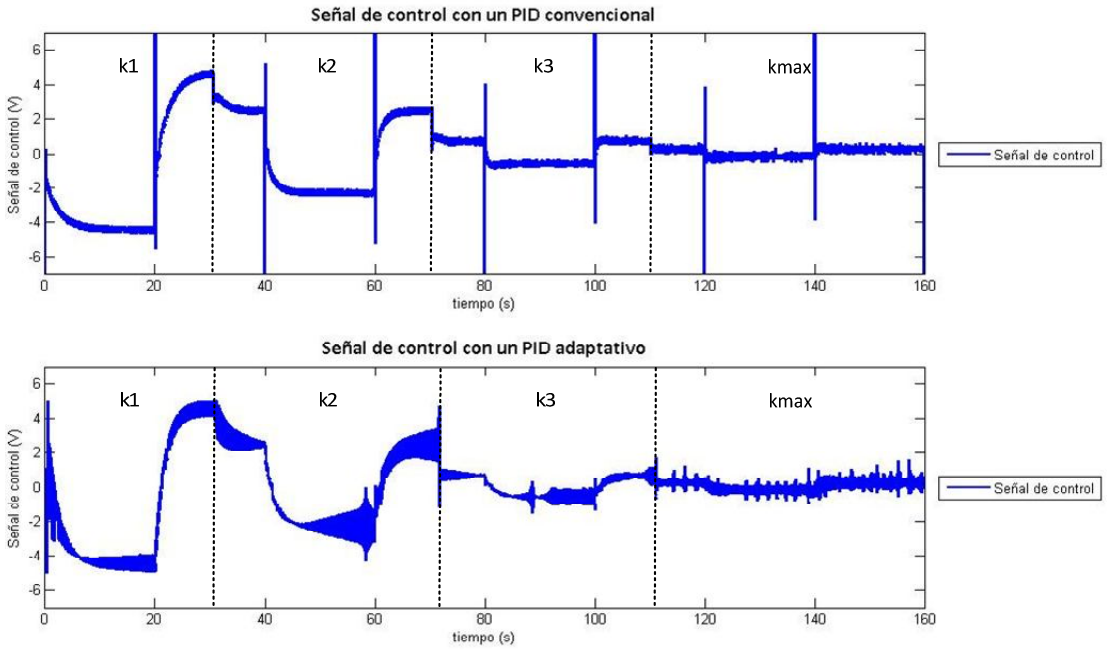

Figura 5.30. Señal de control del control en velocidad con un regulador PID convencional y con un regulador PID adaptativo con k variable

Debido a que los reguladores propuestos se dividen en un regulador convencional que se ejecuta en los momentos iniciales y un regulador adaptativo, además de otros problemas a resolver como el desconocimiento de la planta en el arranque del sistema, el trabajado de diseño se complica respecto al de un regulador convencional de parámetros fijos, lo que debe hacer que se platee cuando realmente son necesarios los reguladores adaptativos. Sin embargo observando los resultados de los ensayos realizados se puede ver que existe una necesidad real en el uso de controladores adaptativos cuando se requiere un control exacto que se ajuste a la dinámica deseada, independientemente de la variación que puedan tener sus parámetros, por lo que sin cuestionar lo acertado o no de la decisión, se ve claramente que según aumenta la precisión requerida también aumenta la necesidad del uso del control adaptativo haciéndose por tanto este tipo control imprescindible.

En el siguiente capítulo se profundizará más en las conclusiones obtenidas de los ensayos que se han realizado para los distintos controladores propuestos.

Capítulo 6. Conclusiones y trabajos futuros

En este capítulo se muestran las conclusiones obtenidas de los distintos ensayos experimentales realizados haciéndose uso tanto de los reguladores adaptativos como del regulador PID convencional y de los trabajos futuros con el fin de conseguir mejoras y explorar otras vías de control adaptativo.

6.1. Conclusiones

Como primera conclusión de gran relevancia, se puede afirmar que es posible la implementación de controladores adaptativos haciendo uso del entorno de Simulink para controlar sistemas físicos en tiempo real, con lo que el objetivo primordial de este proyecto se ha podido alcanzar con unos resultados en general bastantes buenos y más si se tiene en cuenta que la información actual en este campo concreto, y haciendo uso de las mismas herramientas es bastante escasa.

Las conclusiones alcanzadas con los distintos ensayos son las siguientes:

- Es posible la obtención de reguladores adaptativos globalmente estables capaces de trabajar con sistemas físicos haciendo uso de Simulink.
- Los parámetros obtenidos durante la identificación tienden a estabilizarse con el paso del tiempo.
- Existe una dependencia entre el regulador convencional inicial y el adaptativo.
- El regulador PID adaptativo no tiende a inestabilizarse, ni durante la inicialización del sistema ni durante la ejecución. Experimentalmente este regulador es estable.
- El comportamiento en la dinámica adaptativa del regulador PID no tiene gran dependencia del regulador convencional que se ejecuta al arranque.
- El funcionamiento y las señales de control del regulador PID Adaptativo permite que sea usado en entornos reales.
- El controlador Dead-Beat Adaptativo tiende a la inestabilidad en el arranque del sistema, sobre todo si el control se realiza en posición.
- En el controlador Dead-Beat, el comportamiento de la dinámica del regulador adaptativo tiene una gran dependencia del regulador convencional que se ejecuta en la inicialización del sistema. Una mala elección de estos parámetros conlleva un mal control adaptativo.
- La señal de control del controlador Dead-Beat adaptativo hace que este regulador no sea aplicables a sistemas físicos de entornos de trabajo reales.

En el caso de ambos controladores la dinámica obtenida en función de la señal de referencia para las distintas k que se le han dado al motor, han sido las mismas o muy similares, con lo que los controladores adaptativos han sido capaces de asumir estos cambios no solamente fijando una k inicial para todo el ensayo, si no haciendo cambios bruscos de k durante la ejecución, por lo que se obtiene un control adaptativo en tiempo real.

Bien es cierto que el comportamiento del control adaptativo se ve afectado en gran medida por el controlador convencional que actúa en la inicialización del sistema, ya que un número bajo de muestras ejecutándose provoca que la identificación no sea correcta con lo que no se consigue la estabilización posterior con el control adaptativo y un número alto influye demasiado en el comportamiento del sistema en el tiempo. Esto obliga a llegar a un compromiso concreto con cada regulador que haga que el comportamiento inicial no afecte al permanente en exceso, pero que sea suficiente como para que se tenga una buena aproximación con la que el controlador adaptativo pueda adaptar al sistema a la dinámica deseada.

Otro punto importante es que para conseguir estos arranques no solo se necesita ajustar el tiempo, sino también los parámetros del regulador convencional viéndose efectos parecidos a los que se observaban con el tiempo. Si se usan parámetros que fuercen la estabilidad del sistema puede haber efectos negativos de control en el tiempo o si se usan parámetros menos restrictivos ocurre que no se consigue la estabilidad inicial. En el regulador PID este problema concreto es menos perceptible, pero esto no ocurre en el controlador Dead-Beat, principalmente cuando se realiza el control en posición, pudiéndose observar que si se hace uso de unos parámetros muy restrictivos que fuercen al sistema a estabilizarse, al entrar en funcionamiento el control adaptativo no es capaz de conseguir una dinámica correcta, pero si no es así, durante el cambio entre los reguladores el sistema se inestabiliza. La única solución para el caso concreto del controlador Dead-Beat actuando en posición que se puede dar, es realizar arranques con ganancias k del motor pequeñas que permitan un inicio estable y pasado un pequeño periodo de tiempo, que no llega a segundos, reajustar esta k a cualquier valor, obteniéndose un control adaptativo correcto a partir de este momento.

En resumen se puede afirmar que ambos reguladores consiguen la estabilidad global del sistema ajustando la dinámica en función de los distintos parámetros de la planta. En el caso concreto del regulador PID adaptativo los resultados obtenidos son muy buenos, no tiene tendencia a inestabilizarse y el comportamiento en la dinámica del regulador adaptativo no tiene una gran dependencia del regulador convencional.

En el controlador Dead-Beat se vuelve a ver que es globalmente estable y es que es capaz de ajustarse a la dinámica deseada en función de las distintas ganancias k del motor, sin embargo tiende más a la inestabilidad en los primeros periodos de tiempo y la dependencia del controlador adaptativo respecto al convencional es mucho mayor, sobre todo en controles en posición, aunque la principal problemática de este controlador se encuentra en su uso en sistemas reales debido a su señal de control, la cual es de tipo “Todo o nada” con frecuencias

altas, lo que hace que en la mayoría de casos no sea posible su implantación y en caso de serlo puede reducir de forma general la vida del sistema a controlar.

Quizás la cuestión más importante es si realmente los reguladores concretos usados para controlar la dinámica de la maqueta del motor de corriente continua son interesantes o no para implementar en procesos reales al suponer una mejora de control, o si el control convencional sería más que suficiente para estos casos. Si se observan los resultados obtenidos para el controlador PID convencional de parámetros fijos frente al PID adaptativo que se encuentran en el ***Apartado 5.4. Comparativa del control convencional frente al adaptativo***, se puede ver que aunque este regulador se ajuste al régimen permanente, no lo hace en el transitorio, por lo que resultan distintas dinámicas con distintos valores de la ganancia k. Esto implica que en sistemas con gran variabilidad en los parámetros, con un regulador convencional no se tiene un control real sobre la dinámica de este, por lo que es imprescindible el uso de reguladores adaptativos que sean capaces de ajustarse a estos cambios.

6.2. Trabajos futuros

En este apartado se va a intentar dar una idea de ciertos aspectos que podrían ser mejorados con el fin de obtenerse controladores más robustos, en muchos casos, consiguiendo que estas mejoras no solo afecten a los controladores que se han desarrollado sino también a otros que se podrían crear utilizando estos como base, o usando partes de los esquemas de control de los que se disponen. También se quiere mostrar otras líneas de trabajos encaminados a diversos métodos de identificación y control.

6.2.1. Estudio y optimización del uso del bloque S-Function Builder

Al hacerse uso de este bloque ha surgido un problema y es que para el cálculo de los estados discretos actuales pueden utilizarse estados discretos pasados, pero no se puede hacer uso de los valores actuales de las entradas, lo que obliga a usar tiempos de muestreo pequeños para hacer que el error cometido al usar el estado pasado en lugar del actual sea lo más pequeño posible. Este comportamiento es claramente visible al cambiarse los tiempos de muestreo de valores grandes a pequeños durante ensayos de laboratorio, provocando que sistemas regulados que en un principio parecían inestables comenzaran a comportase dinámicamente según se habían diseñado en función se reducía este tiempo.

Aunque la realidad es que esto no es del todo verdad, pues el builder de la S-Function si permite el uso de entradas actuales para el cálculo de salidas actuales e incluso el cálculo estados actuales a partir de las entradas actuales, como muestra la opción que enmarca el recuadro de la ***Figura 6.1***, lo que ocurre es que el cálculo de los estados discretos los realiza después del cálculo de las salidas.

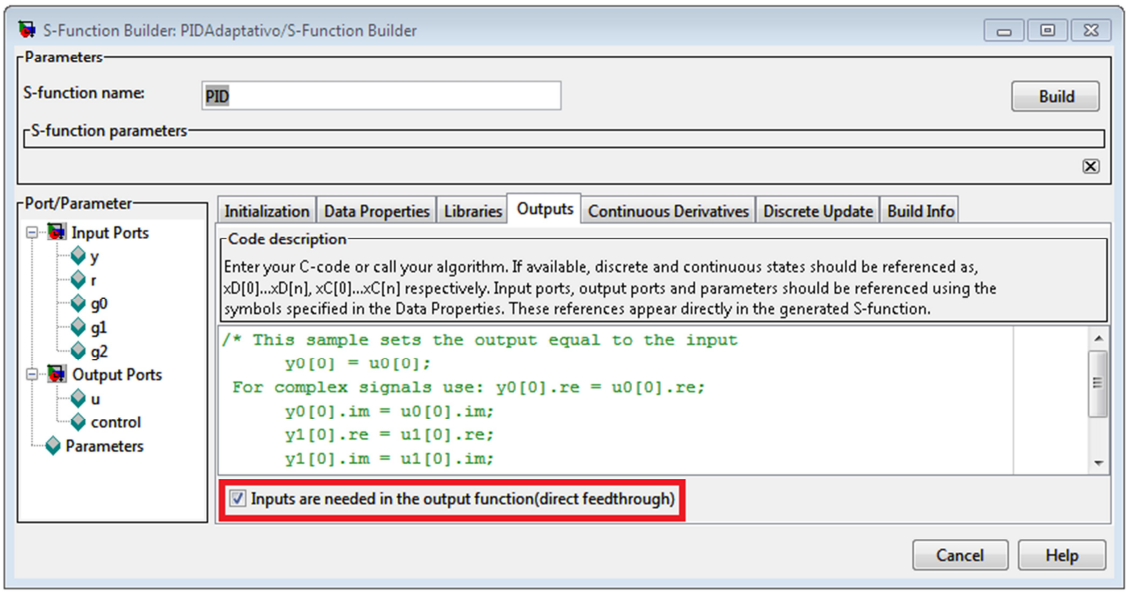

Figura 6.1. Cálculo de la salida del Bloque S-Function Builder.

Teniéndose en cuenta las etapas de ejecución que se vieron en el ***Apartado 3.3.3. S-Function***, para el caso concreto de la implementación de un regulador como los que se han usado, cuya expresión se debe representar en ecuaciones en diferencias con una forma similar a la ecuación ***(6.1)***, el problema se puede evitar dividiendo la ecuación en dos, donde una es la salida que depende de la entrada actual y de un estado discreto pasado, ecuación ***(6.2)***, y la otra el estado discreto que será usado en la siguiente interacción como estado pasado, ecuación ***(6.3)***.

$$y(k) = a \cdot u(k) + b \cdot u(k-1) + c \cdot u(k-2) - d \cdot y(k-1) - e \cdot y(k-2) \quad \textbf{(6.1)}$$

$$y(k) = a \cdot u(k) + xD(k-1) \quad \textbf{(6.2)}$$

$$xD(k) = b \cdot u(k) + c \cdot u(k-1) - d \cdot y(k) - e \cdot y(k-1) \quad \textbf{(6.3)}$$

En el caso concreto de los mínimos cuadrados no es una solución que pudiera ser tan obvia, y es interesante rediseñarlo, no solo por el hecho de tener un control sobre el tiempo de muestreo más acorde con la dinámica de la planta, sino porque además se está introduciendo un error en el sistema que hace que el comportamiento, en mayor o menor medida, empeore.

Otro de los problemas que surgieron fue que los Builder de las S-Function no permiten el enmascaramiento directo y solo se puede enmascarar si antes se integran en un submodelo, es decir, realmente se crea una máscara del submodelo dentro del cual se haya la S-Funcion. En un principio, a pesar de la perdida de configuración y control que esto podría conllevar, no se realizó porque la creación de estos submodelos podrían hacer perder el concepto que se intenta trasmitir con el proyecto, la posibilidad de creación de controles adaptativos con Simulink en tiempo real, sin embargo, en un entorno real y no tan experimental como el que se ha llevado, el estudio de esta opción puede suponer grandes ventajas.

6.2.2. Identificación del sistema en línea

Para la identificación del sistema se ha utilizado el método de mínimos cuadrados recursivo, pero este no es el único que se puede utilizar, existiendo otras vías interesantes de identificación. Además hay una serie de detalles que deben ser atendidos para la mejora del método de mínimos cuadrados recursivo de cara a su optimización.

Mínimos cuadrados recursivo:

Para el proyecto se optó por unos mínimos cuadrados recursivo con la opción de poder ser utilizados con un factor de olvido ajustable, para lo que se hizo uso de un generador de estados diseñado por MATLAB. El uso de este generador de estados trae aparejadas una serie de ventajas que ya se describieron en el ***Apartado 4.2.2. Identificación de procesos en línea***, sin embargo su uso puede hacer que se pierda el nivel de control que se llegaría a conseguir con la integración de este dentro del bloque de mínimos cuadrados. Esta opción unida a las mejoras del ***Apartado 6.2.1. Estudio y optimización del uso del bloque S-Function Builder*** pueden conseguir un método de mínimos cuadrados más robusto y fiable.

Algoritmos de raíz cuadrada:

Aunque estos algoritmos de raíz cuadrada *[2]* no dejan de ser otra variante de los mínimos cuadrados, lo que se pretende conseguir con ellos es una mejora de las propiedades numéricas del método.

La ventaja principal es su doble precisión efectiva respecto a métodos recursivos convencionales, teniendo que realizar para esto un pequeño número de operaciones extra. Esta propiedad es crucial para la identificación en línea, donde errores acumulados durante las muestras de tiempo puede producir que el método no converja.

Algunas de las variantes del algoritmo que se pueden estudiar utilizar para la identificación son:

- REDCO
- REFIL para sistemas discretos lineales
- REFIL para la identificación simultánea de modelos de distinto orden.
- LDFIL
- Descomposición en valores singulares (SVD)

Algoritmos Genéticos:

Los algoritmos genéticos *[21]* tienen su origen en métodos de búsqueda basados en mecanismos de selección natural. La utilidad de estos métodos no solo radica en la posibilidad de saber cuál de los genes de una población tiene más posibilidades de pasar a la siguiente generación o cual tienen más posibilidades de perdurar en el tiempo, sino que además este algoritmo puede permitir la optimización de parámetros no ligados a temas evolutivos proporcionando una poderosa herramienta de optimización de funciones en general.

De esta forma se pueden utilizar los algoritmos genéticos en identificación de sistemas obteniéndose todas las ventajas que traen aparejados o incluso para diseño de controladores *[24]*.

6.2.3. Reguladores adaptativos

La posibilidad de los reguladores adaptativos en la actualidad es mucho más amplia que la se ha visto durante este proyecto, dando otras opciones de control. Para sistemas de control hay dos grandes grupos *[2]*:

1. Sistemas adaptativos por modelo de referencia (MRAS – Model Reference Adaptative System).
2. Reguladores autoajustables (STR – Self-Tunning Regulators). Sería la clasificación de los utilizados en el proyecto y los más extendidos.

Algunos de los existentes en estos dos grupos que son interesantes estudiar se describen a continuación.

Sistemas adaptativos por modelo de referencia (MRAS)

Los MRAS *[2]* utilizan un modelo de referencia que ha sido previamente fijado por el diseñador y que contendrá el comportamiento deseado para el sistema, pudiéndose alcanzar con él un control multivariable. Como se puede ver en la ***Figura 6.2***, a partir de la salida de este modelo de referencia y la de la planta a regular se halla el error entre ambas, siendo esta la base para calcular los parámetros del regulador adaptativo e intentándose en todo momento minimizar este error.

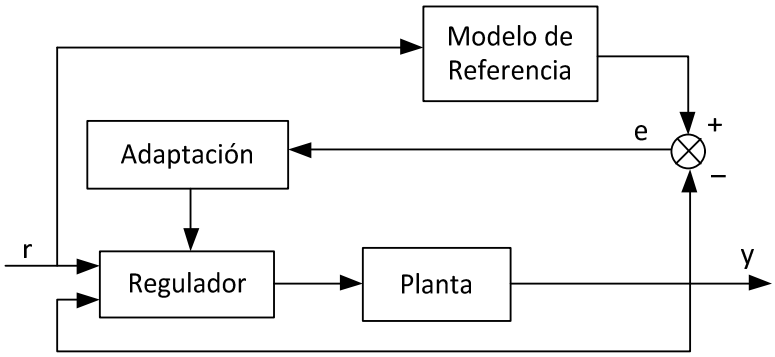

Figura 6.2. Esquema de control de sistemas adaptativos por modelo de referencia.

Con ellos se pueden conseguir resultados más que aceptables en sistemas *[22]* donde sus parámetros tienen una deriva o cambio temporal. Un claro ejemplo está en entornos industriales donde los parámetros de la planta dependerán de la vida que esta misma haya tenido. Para el modelo de referencia escogido hay que tener en cuenta la dinámica de la planta que se quiere controlar, ya que la elección de un modelo con una dinámica muy superior provocará señales de control que pueden llegar a saturarla.

Reguladores autoajustables multivariables

De forma más extendida los reguladores autoajustables *[2]* hacen uso del criterio de mínima varianza en el que se utiliza una ley de control que minimiza la varianza de salida. El problema radica en que este criterio aplicado al ámbito multivariable puede no ser suficiente, además de que trasladar el concepto de regulador de una variable a multivariable no es un trabajo sencillo.

Aun así, existen metodologías para conseguir este control a través de espacios de estados o pseudoestados con los que, aunque se pierde en sencillez respecto al autoajuste de los sistemas de una variable, se gana en funcionalidad. Sin embargo estos métodos de control se hacen indispensables para este tipo de sistemas, por lo que su estudio y aplicación se hace imprescindible.

6.2.4. Estudios de arranque del sistema

Durante el arranque del sistema con los controladores adaptativos implementados se plantea una cuestión difícil, decidir cual deben ser los valores del controlador convencional que deberá controlar el sistema en los primeros instantes de tiempo y durante cuantas muestras de tiempo deben actuar. Esto se produce porque se desconoce cómo va a arrancar ese sistema y aunque se tenga una idea inicial de cómo podría ser, la realidad es que no se sabe a ciencia exacta.

Esto obliga a conseguir la estabilidad del conjunto a toda costa con el fin de que el regulador adaptativo comience a trabajar en una zona estable donde sea capaz de controlar el sistema y adaptarle a la dinámica deseada. Esto es relativamente sencillo de alcanzar, pero puede producir que la dinámica posterior varíe en función de este primer control. Bien es cierto que dependerá tanto del control como del método de identificación utilizado. Un claro ejemplo está en el uso del Método de mínimos cuadrados, ya que cuantas más muestras recordemos, mayor será la dependencia en el tiempo de este primer control.

Saber con exactitud cómo va a reaccionar el conjunto es vital para conseguir un control óptimo, sobre todo en el cambio entre controladores, y puesto que las variantes son elevadas la necesidad de un estudio en profundidad que marque estas pautas de comportamiento es interesante.

6.2.5. Métodos de calibración inicial

Disponer de una herramienta de calibración inicial para los controladores adaptativos podría conseguir no solo mejores arranques del sistema sino la portabilidad de los controladores a otras plantas físicas sin necesidad de realizar ningún tipo de cambio en el modelo. En el caso concreto del proyecto, si los controladores se utilizasen para un motor de corriente continua podrían funcionar o no en función de si los parámetros del nuevo motor distasen mucho de aquellos para el que se realizaron los ensayos, debido a que si la diferencia de estos parámetros es muy grande no se obtendrán inicializaciones al arranque del sistema correctas.

De esta forma bajo una serie de criterios se podría alcanzar la posibilidad de trabajar con un rango más amplio de plantas sin necesidad de realizar ningún tipo de cambio en el controlador, simplemente con un calibrado inicial que proporcionase unos valores aceptables al regulador convencional para que fuera capaz de mantener estable el sistema.

Referencias

[1] *"Evolución histórica de la ingeniería de control" (http://automata.cps.unizar.es/Historia/Webs/IntroduccionI.htm). Fecha de consulta: 23 de septiembre del 2012.*

[2] *Alberto Aguado Behar, Miguel Martínez Iranzo. "Identificación y Control Adaptativo. Robótica y Automática". Pearson Educación. 2002.*

[3] *Práctica 5. "Identificación por mínimos cuadrados recursivo". Modelado e identificación de sistemas, 5º ingeniería industrial. (http://isa.umh.es/asignaturas/mis/ident2.pdf). Fecha de consulta: 23 de septiembre del 2012.*

[4] *Luis Moreno, Santiago Garrido y Carlos Balaguer. "Ingeniería de Control. Modelado y Control de sistemas dinámicos". Ariel Ciencia. 2003.*

[5] *Daniel Rodríguez Ramírez y Carlos Bordons Alba, Depto. de Ingeniería de Sistemas y Automática. "Apuntes de ingeniería de control, análisis y control de sistemas en espacio de estado, identificación de sistemas control adaptativo, control predictivo". 2005.*

[6] *Francisco Salavert Torres. Ingeniería Técnica en Informática de sistemas. "Proyecto Final de Carrera: Implementación de un Regulador PID en un dsPIC33F". 2010.*

[7] *Universidad Carlos III de Madrid. "Curso de especialización: Control inteligente de procesos industriales". 1997.*

[8] *"Capítulo 6. Diseño de control por establecimiento finito (Dead-Beat)" (http://www.kostenlose-bucher.eu/doc/331793/control-ad-or-de-tiempo-finito-dead-beat). Fecha de consulta: 18 de agosto del 2012.*

[9] *Universidad de Buenos aires. "Clase Mínimos Cuadrados. Identificación y Control Adaptativo" (http://materias.fi.uba.ar/6631/material/Clase_05_Minimos_Cuadrados.pdf). Fecha de consulta: 23 de septiembre del 2012.*

[10] *Matlab v7.14.0.739. "Matlab Quick Start".*

[11] *Matlab v7.14.0.739. "Simulink Getting Started, Introduction to Simulink".*

[12] *Matlab v7.14.0.739. "Real-Time Windows Target Getting Started".*

[13] *Matlab v7.14.0.739. "Simulink Normal Mode".*

[14] *Matlab v7.14.0.739. "Simulink External Mode".*

[15] *Matlab v7.14.0.739. "User-Defined Functions".*

[16] Matlab v7.14.0.739. "S-Function Builder".

[17] Matlab v7.14.0.739. "Developing S-Function".

[18] Matlab v7.14.0.739. "How the S-Function Builder Builds an S-Function".

[19] Matlab v7.14.0.739. "Target Language Compiler".

[20] Matlab v7.14.0.739. "TLC Files".

[21] Patricia Gomez Otero. "Estudio de algoritmo genéticos para el desarrollo de sistemas de control".

[22] "Control adaptativo con modelo de referencia". (http://www.slideshare.net/xMorfe0x/control-adaptativo-con-modelo-de-referencia). Fecha de consulta: 23 de septiembre del 2012.

[23] Janeth Carolina Godoy ortega, Escuela politécnica nacional. "Control adaptativo en tiempo real". 2011.

[24] Ilber A. Ruge, Miguel A. Alvis. "Aplicación de los algoritmos genéticos para para el diseño de un PID adaptativo". 2009.

[25] "Introducción al control adaptativo". (http://www.slideshare.net/balzasbravas/control-adaptativo). Fecha de consulta: 23 de septiembre del 2012.

[26] Eugenio Tacconi, Ricardo Mantz, Jorge Solsona y Pablo Puleston, LEICI, Facultad de Ingeniería, UNLP. "Controladores Basados en Estrategias PID". 2005.

Anexos

A. Código de los archivos PID.c y PID_wrapper

Los siguientes archivos son generados por el Builder de la S-Función y contienen el código del regulador PID adaptativo, siendo PID.c una S-Function y PID_wrapper.c un contenedor de una S-Function para mejorar la compatibilidad con el Simulink Coder. Cualquier cambio que se haga en estas funciones a mano, solo se deberá hacer entre los campos:

```
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
```

PID.c

```
/*
 * File: PID1.c
 *
 *
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is an S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_defines_Changes_BEGIN
  *        #define NAME 'replacement text'
  *        %%% SFUNWIZ_defines_Changes_END
  *
  *   DO NOT change NAME--Change the 'replacement text' only.
  *
  *   For better compatibility with the Simulink Coder, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Simulink Coder's Manual in the Chapter titled,
  *   "Wrapper S-functions".
  *
  *  -------------------------------------------------------------------------
-
  * | See matlabroot/simulink/src/sfuntmpl_doc.c for a more detailed template
|
  *  -------------------------------------------------------------------------
-
 * Created: Wed Jul 18 17:53:18 2012
 *
 *
 */
```

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME PID1

/*<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*/
/* %%%-SFUNWIZ_defines_Changes_BEGIN --- EDIT HERE TO _END */
#define NUM_INPUTS          4

/* Input Port  0 */
#define IN_PORT_0_NAME      r
#define INPUT_0_WIDTH       1
#define INPUT_DIMS_0_COL    1
#define INPUT_0_DTYPE       real_T
#define INPUT_0_COMPLEX     COMPLEX_NO
#define IN_0_FRAME_BASED    FRAME_NO
#define IN_0_BUS_BASED      0
#define IN_0_BUS_NAME
#define IN_0_DIMS           1-D
#define INPUT_0_FEEDTHROUGH 1
#define IN_0_ISSIGNED        0
#define IN_0_WORDLENGTH      8
#define IN_0_FIXPOINTSCALING 1
#define IN_0_FRACTIONLENGTH  9
#define IN_0_BIAS            0
#define IN_0_SLOPE           0.125

/* Input Port  1 */
#define IN_PORT_1_NAME      g0
#define INPUT_1_WIDTH       1
#define INPUT_DIMS_1_COL    1
#define INPUT_1_DTYPE       real_T
#define INPUT_1_COMPLEX     COMPLEX_NO
#define IN_1_FRAME_BASED    FRAME_NO
#define IN_1_BUS_BASED      0
#define IN_1_BUS_NAME
#define IN_1_DIMS           1-D
#define INPUT_1_FEEDTHROUGH 1
#define IN_1_ISSIGNED        0
#define IN_1_WORDLENGTH      8
#define IN_1_FIXPOINTSCALING 1
#define IN_1_FRACTIONLENGTH  9
#define IN_1_BIAS            0
#define IN_1_SLOPE           0.125

/* Input Port  2 */
#define IN_PORT_2_NAME      g1
#define INPUT_2_WIDTH       1
#define INPUT_DIMS_2_COL    1
#define INPUT_2_DTYPE       real_T
#define INPUT_2_COMPLEX     COMPLEX_NO
#define IN_2_FRAME_BASED    FRAME_NO
#define IN_2_BUS_BASED      0
#define IN_2_BUS_NAME
#define IN_2_DIMS           1-D
#define INPUT_2_FEEDTHROUGH 1
#define IN_2_ISSIGNED        0
#define IN_2_WORDLENGTH      8
#define IN_2_FIXPOINTSCALING 1
#define IN_2_FRACTIONLENGTH  9
#define IN_2_BIAS            0
#define IN_2_SLOPE           0.125
```

```
/* Input Port  3 */
#define IN_PORT_3_NAME      g2
#define INPUT_3_WIDTH       1
#define INPUT_DIMS_3_COL    1
#define INPUT_3_DTYPE       real_T
#define INPUT_3_COMPLEX     COMPLEX_NO
#define IN_3_FRAME_BASED    FRAME_NO
#define IN_3_BUS_BASED      0
#define IN_3_BUS_NAME
#define IN_3_DIMS           1-D
#define INPUT_3_FEEDTHROUGH 1
#define IN_3_ISSIGNED        0
#define IN_3_WORDLENGTH      8
#define IN_3_FIXPOINTSCALING 1
#define IN_3_FRACTIONLENGTH  9
#define IN_3_BIAS            0
#define IN_3_SLOPE           0.125

#define NUM_OUTPUTS          2

/* Output Port  0 */
#define OUT_PORT_0_NAME      u1
#define OUTPUT_0_WIDTH       1
#define OUTPUT_DIMS_0_COL    1
#define OUTPUT_0_DTYPE       real_T
#define OUTPUT_0_COMPLEX     COMPLEX_NO
#define OUT_0_FRAME_BASED    FRAME_NO
#define OUT_0_BUS_BASED      0
#define OUT_0_BUS_NAME
#define OUT_0_DIMS           1-D
#define OUT_0_ISSIGNED        1
#define OUT_0_WORDLENGTH      8
#define OUT_0_FIXPOINTSCALING 1
#define OUT_0_FRACTIONLENGTH  3
#define OUT_0_BIAS            0
#define OUT_0_SLOPE           0.125

/* Output Port  1 */
#define OUT_PORT_1_NAME      control
#define OUTPUT_1_WIDTH       1
#define OUTPUT_DIMS_1_COL    1
#define OUTPUT_1_DTYPE       real_T
#define OUTPUT_1_COMPLEX     COMPLEX_NO
#define OUT_1_FRAME_BASED    FRAME_NO
#define OUT_1_BUS_BASED      0
#define OUT_1_BUS_NAME
#define OUT_1_DIMS           1-D
#define OUT_1_ISSIGNED        1
#define OUT_1_WORDLENGTH      8
#define OUT_1_FIXPOINTSCALING 1
#define OUT_1_FRACTIONLENGTH  3
#define OUT_1_BIAS            0
#define OUT_1_SLOPE           0.125

#define NPARAMS              0

#define SAMPLE_TIME_0        0.001
#define NUM_DISC_STATES      5
#define DISC_STATES_IC       [0,0,0,0,0]
#define NUM_CONT_STATES      0
#define CONT_STATES_IC       [0]
```

```
#define SFUNWIZ_GENERATE_TLC 1
#define SOURCEFILES "__SFB__"
#define PANELINDEX           6
#define USE_SIMSTRUCT        0
#define SHOW_COMPILE_STEPS   0
#define CREATE_DEBUG_MEXFILE 0
#define SAVE_CODE_ONLY       0
#define SFUNWIZ_REVISION     3.0
/* %%%-SFUNWIZ_defines_Changes_END --- EDIT HERE TO _BEGIN */
/*<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*/

#include "simstruc.h"

extern void PID1_Outputs_wrapper(const real_T *r,
                          const real_T *g0,
                          const real_T *g1,
                          const real_T *g2,
                          real_T *u1,
                          real_T *control,
                          const real_T  *xD);
extern void PID1_Update_wrapper(const real_T *r,
                          const real_T *g0,
                          const real_T *g1,
                          const real_T *g2,
                          const real_T *u1,
                          const real_T *control,
                          real_T *xD);

/*====================*
 * S-function methods *
 *====================*/
/*                          Function:                          mdlInitializeSizes
=============================================
 * Abstract:
 *   Setup sizes of the various vectors.
 */

static void mdlInitializeSizes(SimStruct *S)
{

    DECL_AND_INIT_DIMSINFO(inputDimsInfo);
    DECL_AND_INIT_DIMSINFO(outputDimsInfo);
    ssSetNumSFcnParams(S, NPARAMS);
     if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
     return; /* Parameter mismatch will be reported by Simulink */
     }

    ssSetNumContStates(S, NUM_CONT_STATES);
    ssSetNumDiscStates(S, NUM_DISC_STATES);

    if (!ssSetNumInputPorts(S, NUM_INPUTS)) return;

    /*Input Port 0 */
    ssSetInputPortWidth(S,  0, INPUT_0_WIDTH); /* */
    ssSetInputPortDataType(S, 0, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  0, INPUT_0_COMPLEX);
    ssSetInputPortDirectFeedThrough(S, 0, INPUT_0_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 0, 1); /*direct input signal access*/
```

```
    /*Input Port 1 */
    ssSetInputPortWidth(S,  1, INPUT_1_WIDTH); /* */
    ssSetInputPortDataType(S, 1, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  1, INPUT_1_COMPLEX);
    ssSetInputPortDirectFeedThrough(S, 1, INPUT_1_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 1, 1); /*direct input signal access*/

    /*Input Port 2 */
    ssSetInputPortWidth(S,  2, INPUT_2_WIDTH); /* */
    ssSetInputPortDataType(S, 2, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  2, INPUT_2_COMPLEX);
    ssSetInputPortDirectFeedThrough(S, 2, INPUT_2_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 2, 1); /*direct input signal access*/

    /*Input Port 3 */
    ssSetInputPortWidth(S,  3, INPUT_3_WIDTH); /* */
    ssSetInputPortDataType(S, 3, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  3, INPUT_3_COMPLEX);
    ssSetInputPortDirectFeedThrough(S, 3, INPUT_3_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 3, 1); /*direct input signal access*/


    if (!ssSetNumOutputPorts(S, NUM_OUTPUTS)) return;

    /* Output Port 0 */
    ssSetOutputPortWidth(S, 0, OUTPUT_0_WIDTH);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 0, OUTPUT_0_COMPLEX);

    /* Output Port 1 */
    ssSetOutputPortWidth(S, 1, OUTPUT_1_WIDTH);
    ssSetOutputPortDataType(S, 1, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 1, OUTPUT_1_COMPLEX);

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    /* Take care when specifying exception free code - see sfuntmpl_doc.c */
    ssSetOptions(S, (SS_OPTION_EXCEPTION_FREE_CODE |
                     SS_OPTION_USE_TLC_WITH_ACCELERATOR |
             SS_OPTION_WORKS_WITH_CODE_REUSE));
}

# define MDL_SET_INPUT_PORT_FRAME_DATA
static void mdlSetInputPortFrameData(SimStruct  *S,
                                     int_T      port,
                                     Frame_T    frameData)
{
    ssSetInputPortFrameData(S, port, frameData);
}

/*                     Function:                     mdlInitializeSampleTimes
=========================================
 * Abstract:
 *    Specifiy  the sample time.
```

```
 */

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, SAMPLE_TIME_0);
    ssSetOffsetTime(S, 0, 0.0);
}
#define MDL_INITIALIZE_CONDITIONS

 /*                     Function:                     mdlInitializeConditions
========================================
  * Abstract:
  *    Initialize the states
  */

 static void mdlInitializeConditions(SimStruct *S)
 {
   real_T *xD   = ssGetRealDiscStates(S);


    xD[0] =  0;
    xD[1] =  0;
    xD[2] =  0;
    xD[3] =  0;
    xD[4] =  0;

 }
#define MDL_SET_INPUT_PORT_DATA_TYPE
static void mdlSetInputPortDataType(SimStruct *S, int port, DTypeId dType)
{
    ssSetInputPortDataType( S, 0, dType);
}
#define MDL_SET_OUTPUT_PORT_DATA_TYPE
static void mdlSetOutputPortDataType(SimStruct *S, int port, DTypeId dType)
{
    ssSetOutputPortDataType(S, 0, dType);
}

#define MDL_SET_DEFAULT_PORT_DATA_TYPES
static void mdlSetDefaultPortDataTypes(SimStruct *S)
{
  ssSetInputPortDataType( S, 0, SS_DOUBLE);
 ssSetOutputPortDataType(S, 0, SS_DOUBLE);
}

/*                            Function:                            mdlOutputs
=======================================================
 *
*/

static void mdlOutputs(SimStruct *S, int_T tid)
{
    const real_T   *r  = (const real_T*) ssGetInputPortSignal(S,0);
    const real_T   *g0  = (const real_T*) ssGetInputPortSignal(S,1);
    const real_T   *g1  = (const real_T*) ssGetInputPortSignal(S,2);
    const real_T   *g2  = (const real_T*) ssGetInputPortSignal(S,3);
    real_T        *u1  = (real_T *)ssGetOutputPortRealSignal(S,0);
    real_T        *control  = (real_T *)ssGetOutputPortRealSignal(S,1);
    const real_T   *xD = ssGetDiscStates(S);

    PID1_Outputs_wrapper(r, g0, g1, g2, u1, control, xD);
```

```
}

#define MDL_UPDATE  /* Change to #undef to remove function */
/* Function: mdlUpdate ======================================================
   * Abstract:
   *    This function is called once for every major integration time step.
   *      Discrete states are typically updated here, but this function is
useful
   *    for performing any tasks that should only take place once per
   *    integration step.
   */
  static void mdlUpdate(SimStruct *S, int_T tid)
  {
    real_T         *xD  = ssGetDiscStates(S);
    const real_T   *r  = (const real_T*) ssGetInputPortSignal(S,0);
    const real_T   *g0  = (const real_T*) ssGetInputPortSignal(S,1);
    const real_T   *g1  = (const real_T*) ssGetInputPortSignal(S,2);
    const real_T   *g2  = (const real_T*) ssGetInputPortSignal(S,3);
    real_T        *u1  = (real_T *)ssGetOutputPortRealSignal(S,0);
    real_T        *control  = (real_T *)ssGetOutputPortRealSignal(S,1);

    PID1_Update_wrapper(r, g0, g1, g2, u1, control,  xD);
}


/*                            Function:                            mdlTerminate
=====================================================
 * Abstract:
 *    In this function, you should perform any actions that are necessary
 *    at the termination of a simulation.  For example, if memory was
 *    allocated in mdlStart, this is the place to free it.
 */
static void mdlTerminate(SimStruct *S)
{
}
#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

PID_wrapper.c

```
/*
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is a wrapper S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
  *            Your Changes go here
  *        %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
```

```
  *
  *   For better compatibility with the Simulink Coder, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Simulink Coder User's Manual in the Chapter titled,
  *   "Wrapper S-functions".
  *
  *   Created: Wed Jul 18 17:53:18 2012
  */



/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 1
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
 * Output functions
 *
 */
void PID1_Outputs_wrapper(const real_T *r,
                          const real_T *g0,
                          const real_T *g1,
                          const real_T *g2,
                          real_T *u1,
                          real_T *control,
                          const real_T  *xD)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
u1[0] = xD[0];
control[0]=xD[4];
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
  * Updates function
  *
  */
void PID1_Update_wrapper(const real_T *r,
                          const real_T *g0,
                          const real_T *g1,
                          const real_T *g2,
                          const real_T *u1,
                          const real_T *control,
```

```
                            real_T *xD)
{

  /* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
if ((xD[3]<2000)){

    xD[0] = xD[1]+(1.5862e-04)*xD[2];

    xD[4] = 0;
    xD[3] = xD[3]+1;
 }

else{
    xD[0] = xD[1]+(g0[0]+g1[0]+g2[0])*xD[2];
    xD[4] = 1;
}

if((r[0]==0)&&(xD[4]==0)){
    xD[3] = 0;
}



xD[1] = xD[0];
xD[2] = r[0];
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
}
```

B. Código de los archivos DB1.c y DB1_wrapper

Los siguientes archivos son generados por el Builder de la S-Función y contienen el código del controlador Dead-Beat adaptativo, siendo DB1.c una S-Function y DB1_wrapper.c un contenedor de una S-Function para mejorar la compatibilidad con el Simulink Coder. Cualquier cambio que se haga en estas funciones a mano, solo se deberá hacer entre los campos:

```
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
```

DB1.c

```
/*
 * File: DB1.c
 *
 *
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is an S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_defines_Changes_BEGIN
  *        #define NAME 'replacement text'
  *        %%% SFUNWIZ_defines_Changes_END
  *
  *   DO NOT change NAME--Change the 'replacement text' only.
  *
  *   For better compatibility with the Simulink Coder, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Simulink Coder's Manual in the Chapter titled,
  *   "Wrapper S-functions".
  *
  *  ------------------------------------------------------------------------
-
  * | See matlabroot/simulink/src/sfuntmpl_doc.c for a more detailed template
|
  *  ------------------------------------------------------------------------
-
 * Created: Wed Aug 22 12:30:13 2012
 *
 *
 */

#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME DB1

/*<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*/
/* %%%-SFUNWIZ_defines_Changes_BEGIN --- EDIT HERE TO _END */

#define NUM_INPUTS          2
```

```
/* Input Port  0 */
#define IN_PORT_0_NAME      u
#define INPUT_0_WIDTH       1
#define INPUT_DIMS_0_COL    1
#define INPUT_0_DTYPE       real_T
#define INPUT_0_COMPLEX     COMPLEX_NO
#define IN_0_FRAME_BASED    FRAME_NO
#define IN_0_BUS_BASED      0
#define IN_0_BUS_NAME
#define IN_0_DIMS           1-D
#define INPUT_0_FEEDTHROUGH 1
#define IN_0_ISSIGNED        0
#define IN_0_WORDLENGTH      8
#define IN_0_FIXPOINTSCALING 1
#define IN_0_FRACTIONLENGTH  9
#define IN_0_BIAS            0
#define IN_0_SLOPE           0.125

/* Input Port  1 */
#define IN_PORT_1_NAME      paramDB
#define INPUT_1_WIDTH       5
#define INPUT_DIMS_1_COL    1
#define INPUT_1_DTYPE       real_T
#define INPUT_1_COMPLEX     COMPLEX_NO
#define IN_1_FRAME_BASED    FRAME_NO
#define IN_1_BUS_BASED      0
#define IN_1_BUS_NAME
#define IN_1_DIMS           1-D
#define INPUT_1_FEEDTHROUGH 1
#define IN_1_ISSIGNED        0
#define IN_1_WORDLENGTH      8
#define IN_1_FIXPOINTSCALING 1
#define IN_1_FRACTIONLENGTH  9
#define IN_1_BIAS            0
#define IN_1_SLOPE           0.125

#define NUM_OUTPUTS          2

/* Output Port  0 */
#define OUT_PORT_0_NAME      y
#define OUTPUT_0_WIDTH       1
#define OUTPUT_DIMS_0_COL    1
#define OUTPUT_0_DTYPE       real_T
#define OUTPUT_0_COMPLEX     COMPLEX_NO
#define OUT_0_FRAME_BASED    FRAME_NO
#define OUT_0_BUS_BASED      0
#define OUT_0_BUS_NAME
#define OUT_0_DIMS           1-D
#define OUT_0_ISSIGNED        1
#define OUT_0_WORDLENGTH      8
#define OUT_0_FIXPOINTSCALING 1
#define OUT_0_FRACTIONLENGTH  3
#define OUT_0_BIAS            0
#define OUT_0_SLOPE           0.125

/* Output Port  1 */
#define OUT_PORT_1_NAME      Ctrl
#define OUTPUT_1_WIDTH       1
#define OUTPUT_DIMS_1_COL    1
#define OUTPUT_1_DTYPE       real_T
#define OUTPUT_1_COMPLEX     COMPLEX_NO
```

```
#define OUT_1_FRAME_BASED    FRAME_NO
#define OUT_1_BUS_BASED      0
#define OUT_1_BUS_NAME
#define OUT_1_DIMS           1-D
#define OUT_1_ISSIGNED        1
#define OUT_1_WORDLENGTH      8
#define OUT_1_FIXPOINTSCALING 1
#define OUT_1_FRACTIONLENGTH  3
#define OUT_1_BIAS            0
#define OUT_1_SLOPE           0.125

#define NPARAMS              0

#define SAMPLE_TIME_0        0.001
#define NUM_DISC_STATES      7
#define DISC_STATES_IC       [0,0,0,0,0,0,0]
#define NUM_CONT_STATES      0
#define CONT_STATES_IC       [0]

#define SFUNWIZ_GENERATE_TLC 1
#define SOURCEFILES "__SFB__"
#define PANELINDEX           6
#define USE_SIMSTRUCT        0
#define SHOW_COMPILE_STEPS   0
#define CREATE_DEBUG_MEXFILE 0
#define SAVE_CODE_ONLY       0
#define SFUNWIZ_REVISION     3.0
/* %%%-SFUNWIZ_defines_Changes_END --- EDIT HERE TO _BEGIN */
/*<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*/
#include "simstruc.h"

extern void DB1_Outputs_wrapper(const real_T *u,
                          const real_T *paramDB,
                          real_T *y,
                          real_T *Ctrl,
                          const real_T  *xD);
extern void DB1_Update_wrapper(const real_T *u,
                          const real_T *paramDB,
                          const real_T *y,
                          const real_T *Ctrl,
                          real_T *xD);

/*====================*
 * S-function methods *
 *====================*/
/*                          Function:                          mdlInitializeSizes
================================================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{

    DECL_AND_INIT_DIMSINFO(inputDimsInfo);
    DECL_AND_INIT_DIMSINFO(outputDimsInfo);
    ssSetNumSFcnParams(S, NPARAMS);
     if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
     return; /* Parameter mismatch will be reported by Simulink */
     }

    ssSetNumContStates(S, NUM_CONT_STATES);
    ssSetNumDiscStates(S, NUM_DISC_STATES);
```

```
    if (!ssSetNumInputPorts(S, NUM_INPUTS)) return;
    /*Input Port 0 */
    ssSetInputPortWidth(S,  0, INPUT_0_WIDTH); /* */
    ssSetInputPortDataType(S, 0, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  0, INPUT_0_COMPLEX);
    ssSetInputPortDirectFeedThrough(S, 0, INPUT_0_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 0, 1); /*direct input signal access*/

    /*Input Port 1 */
    ssSetInputPortWidth(S,  1, INPUT_1_WIDTH);
    ssSetInputPortDataType(S, 1, SS_DOUBLE);
    ssSetInputPortComplexSignal(S, 1, INPUT_1_COMPLEX);
    ssSetInputPortDirectFeedThrough(S, 1, INPUT_1_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 1, 1); /*direct input signal access*/


    if (!ssSetNumOutputPorts(S, NUM_OUTPUTS)) return;
    /* Output Port 0 */
    ssSetOutputPortWidth(S, 0, OUTPUT_0_WIDTH);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 0, OUTPUT_0_COMPLEX);
    /* Output Port 1 */
    ssSetOutputPortWidth(S, 1, OUTPUT_1_WIDTH);
    ssSetOutputPortDataType(S, 1, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 1, OUTPUT_1_COMPLEX);

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    /* Take care when specifying exception free code - see sfuntmpl_doc.c */
    ssSetOptions(S, (SS_OPTION_EXCEPTION_FREE_CODE |
                     SS_OPTION_USE_TLC_WITH_ACCELERATOR |
             SS_OPTION_WORKS_WITH_CODE_REUSE));
}

# define MDL_SET_INPUT_PORT_FRAME_DATA
static void mdlSetInputPortFrameData(SimStruct  *S,
                                     int_T      port,
                                     Frame_T    frameData)
{
    ssSetInputPortFrameData(S, port, frameData);
}
/*                     Function:                     mdlInitializeSampleTimes
=========================================
 * Abstract:
 *    Specifiy  the sample time.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, SAMPLE_TIME_0);
    ssSetOffsetTime(S, 0, 0.0);
}
#define MDL_INITIALIZE_CONDITIONS
 /*                      Function:                     mdlInitializeConditions
=========================================
  * Abstract:
  *    Initialize the states
```

```
  */
 static void mdlInitializeConditions(SimStruct *S)
 {
   real_T *xD   = ssGetRealDiscStates(S);

    xD[0] =  0;
    xD[1] =  0;
    xD[2] =  0;
    xD[3] =  0;
    xD[4] =  0;
    xD[5] =  0;
    xD[6] =  0;

 }
#define MDL_SET_INPUT_PORT_DATA_TYPE
static void mdlSetInputPortDataType(SimStruct *S, int port, DTypeId dType)
{
    ssSetInputPortDataType( S, 0, dType);
}
#define MDL_SET_OUTPUT_PORT_DATA_TYPE
static void mdlSetOutputPortDataType(SimStruct *S, int port, DTypeId dType)
{
    ssSetOutputPortDataType(S, 0, dType);
}

#define MDL_SET_DEFAULT_PORT_DATA_TYPES
static void mdlSetDefaultPortDataTypes(SimStruct *S)
{
  ssSetInputPortDataType( S, 0, SS_DOUBLE);
 ssSetOutputPortDataType(S, 0, SS_DOUBLE);
}
/*                              Function:                              mdlOutputs
=============================================================
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    const real_T   *u  = (const real_T*) ssGetInputPortSignal(S,0);
    const real_T   *paramDB  = (const real_T*) ssGetInputPortSignal(S,1);
    real_T        *y  = (real_T *)ssGetOutputPortRealSignal(S,0);
    real_T        *Ctrl  = (real_T *)ssGetOutputPortRealSignal(S,1);
    const real_T   *xD = ssGetDiscStates(S);

    DB1_Outputs_wrapper(u, paramDB, y, Ctrl, xD);
}
#define MDL_UPDATE  /* Change to #undef to remove function */
/* Function: mdlUpdate ======================================================
   * Abstract:
   *    This function is called once for every major integration time step.
   *     Discrete states are typically updated here, but this function is
useful
   *    for performing any tasks that should only take place once per
   *    integration step.
   */
  static void mdlUpdate(SimStruct *S, int_T tid)
  {
    real_T         *xD  = ssGetDiscStates(S);
    const real_T   *u  = (const real_T*) ssGetInputPortSignal(S,0);
    const real_T   *paramDB  = (const real_T*) ssGetInputPortSignal(S,1);
    real_T        *y  = (real_T *)ssGetOutputPortRealSignal(S,0);
    real_T        *Ctrl  = (real_T *)ssGetOutputPortRealSignal(S,1);

    DB1_Update_wrapper(u, paramDB, y, Ctrl,  xD);
```

```
}
/*                              Function:                              mdlTerminate
=======================================================
 * Abstract:
 *    In this function, you should perform any actions that are necessary
 *    at the termination of a simulation.  For example, if memory was
 *    allocated in mdlStart, this is the place to free it.
 */
static void mdlTerminate(SimStruct *S)
{
}
#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

DB1_wrapper.c

```
/*
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is a wrapper S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
  *            Your Changes go here
  *        %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
  *
  *   For better compatibility with the Simulink Coder, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Simulink Coder User's Manual in the Chapter titled,
  *   "Wrapper S-functions".
  *
  *   Created: Wed Aug 22 12:30:13 2012
  */



/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 1
#define y_width 1
```

```
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
 * Output functions
 *
 */
void DB1_Outputs_wrapper(const real_T *u,
                         const real_T *paramDB,
                         real_T *y,
                         real_T *Ctrl,
                         const real_T  *xD)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
y[0] = xD[0];
Ctrl[0] = xD[5];
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
  * Updates function
  *
  */
void DB1_Update_wrapper(const real_T *u,
                         const real_T *paramDB,
                         const real_T *y,
                         const real_T *Ctrl,
                         real_T *xD)
{
  /* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
if ((xD[6]<75)){

    xD[0] = xD[1] + 448.2894*u[0] - 222.9678*xD[3] - 223.3068*xD[4];

    xD[5] = 0;
    xD[6] = xD[6]+1;
}

else{

    xD[0] = paramDB[0]*xD[1] + paramDB[1]*xD[2] + paramDB[2]*u[0] +
            paramDB[3]*xD[3] + paramDB[4]*xD[4];
    xD[5] = 1;

}

if((u[0]==0)&&(xD[5]==0)){
    xD[6] = 0;
}

xD[2] = xD[1];
xD[1] = xD[0];
xD[4] = xD[3];
xD[3] = u[0];
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
```

```
}
```

C. Código de mínimos cuadrados usando un S-Function Builder

El bloque que se puede ver en la *Figura C.1* realiza la función de mínimos cuadrados recursivo y se ha hecho uso para su generación un S-Function Builder.

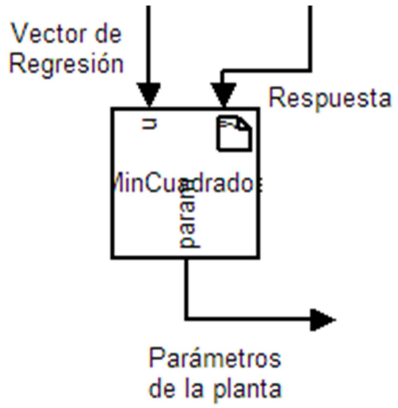

Figura C.1. Bloque de la S-Function de Mínimos Cuadrados Recursivos

Puesto que sería excesivamente extenso, solo se pondrá la información y configuración contenida en el Builder y se obviará los archivos MinCuadrados.c y MinCuadrados_ wrapper.c que este bloque genera de forma automática.

Configuración elegida para el Builder es:

Número de estados discretos	20
Valores iniciales de los ED	0,0,0,0,1e6,0,0,0,0,1e6,0,0,0,0,1e6,0,0,0,0,1e6
Modo de muestra	Discreto
Valor del tiempo de muestra	0.001

Tabla C.1. Valores de inicialización del S-Function Builder del bloque de mínimos cuadrados

El código usado para la salida de la S-Function es la siguiente:

```
param[0]=xD[0];
param[1]=xD[1];
param[2]=xD[2];
param[3]=xD[3];
```

y el código para la actualización de los estados discretos:

```
/*************************/
/* Definición de matrices */
/*************************/

real_T theta[4];
real_T P[4][4];
real_T phi[4];
real_T est_err[1];
real_T theta_New[4];
real_T P_New[4][4];
real_T den[1];
real_T lambda[1];
```

```
/*********************/
/* Montando Matrices. */
/*********************/

theta[0]=xD[0];theta[1]=xD[1];theta[2]=xD[2];theta[3]=xD[3];

P[0][0]=xD[4];  P[0][1]=xD[5];  P[0][2]=xD[6];  P[0][3]=xD[7];
P[1][0]=xD[8];  P[1][1]=xD[9];  P[1][2]=xD[10]; P[1][3]=xD[11];
P[2][0]=xD[12]; P[2][1]=xD[13]; P[2][2]=xD[14]; P[2][3]=xD[15];
P[3][0]=xD[16]; P[3][1]=xD[17]; P[3][2]=xD[18]; P[3][3]=xD[19];

phi[0]=u[0];phi[1]=u[1];phi[2]=u[2];phi[3]=u[3];

lambda[0] = 1;


/*********************/
/* Mínimos Cuadrados. */
/*********************/

est_err[0] = y[0] - phi[0]*theta[0] - phi[1]*theta[1] - phi[2]*theta[2] -
phi[3]*theta[3];


den[0] = lambda[0] + phi[0]*(P[0][0]*phi[0] + P[1][0]*phi[1] + P[2][0]*phi[2]
+ P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] + P[2][1]*phi[2] +
P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] + P[2][2]*phi[2] +
P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] + P[2][3]*phi[2] +
P[3][3]*phi[3]);


theta_New[0]     =     theta[0]     -     ((P[0][0]*phi[0])/(lambda[0]     +
phi[0]*(P[0][0]*phi[0] + P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) +
phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) +
phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) +
phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])) +
(P[0][1]*phi[1])/(lambda[0]   +   phi[0]*(P[0][0]*phi[0]   +   P[1][0]*phi[1]   +
P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] +
P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] +
P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] +
P[2][3]*phi[2]    +    P[3][3]*phi[3]))    +    (P[0][2]*phi[2])/(lambda[0]    +
phi[0]*(P[0][0]*phi[0] + P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) +
phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) +
phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) +
phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])) +
(P[0][3]*phi[3])/(lambda[0]   +   phi[0]*(P[0][0]*phi[0]   +   P[1][0]*phi[1]   +
P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] +
P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] +
P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] +
P[2][3]*phi[2] + P[3][3]*phi[3]))*(phi[0]*theta[0] - y[0] + phi[1]*theta[1] +
phi[2]*theta[2] + phi[3]*theta[3]);

theta_New[1]     =     theta[1]     -     ((P[1][0]*phi[0])/(lambda[0]     +
phi[0]*(P[0][0]*phi[0] + P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) +
phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) +
phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) +
phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])) +
(P[1][1]*phi[1])/(lambda[0]   +   phi[0]*(P[0][0]*phi[0]   +   P[1][0]*phi[1]   +
P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] +
```

```
P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] +
P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] +
P[2][3]*phi[2]    +    P[3][3]*phi[3]))    +    (P[1][2]*phi[2])/(lambda[0]    +
phi[0]*(P[0][0]*phi[0] + P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) +
phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) +
phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) +
phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])) +
(P[1][3]*phi[3])/(lambda[0]   +   phi[0]*(P[0][0]*phi[0]   +   P[1][0]*phi[1]   +
P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] +
P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] +
P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] +
P[2][3]*phi[2] + P[3][3]*phi[3])))*(phi[0]*theta[0] - y[0] + phi[1]*theta[1] +
phi[2]*theta[2] + phi[3]*theta[3]);

theta_New[2]      =      theta[2]      -      ((P[2][0]*phi[0])/(lambda[0]      +
phi[0]*(P[0][0]*phi[0] + P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) +
phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) +
phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) +
phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])) +
(P[2][1]*phi[1])/(lambda[0]   +   phi[0]*(P[0][0]*phi[0]   +   P[1][0]*phi[1]   +
P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] +
P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] +
P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] +
P[2][3]*phi[2]    +    P[3][3]*phi[3]))    +    (P[2][2]*phi[2])/(lambda[0]    +
phi[0]*(P[0][0]*phi[0] + P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) +
phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) +
phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) +
phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])) +
(P[2][3]*phi[3])/(lambda[0]   +   phi[0]*(P[0][0]*phi[0]   +   P[1][0]*phi[1]   +
P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] +
P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] +
P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] +
P[2][3]*phi[2] + P[3][3]*phi[3])))*(phi[0]*theta[0] - y[0] + phi[1]*theta[1] +
phi[2]*theta[2] + phi[3]*theta[3]);

theta_New[3]      =      theta[3]      -      ((P[3][0]*phi[0])/(lambda[0]      +
phi[0]*(P[0][0]*phi[0] + P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) +
phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) +
phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) +
phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])) +
(P[3][1]*phi[1])/(lambda[0]   +   phi[0]*(P[0][0]*phi[0]   +   P[1][0]*phi[1]   +
P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] +
P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] +
P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] +
P[2][3]*phi[2]    +    P[3][3]*phi[3]))    +    (P[3][2]*phi[2])/(lambda[0]    +
phi[0]*(P[0][0]*phi[0] + P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) +
phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) +
phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) +
phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])) +
(P[3][3]*phi[3])/(lambda[0]   +   phi[0]*(P[0][0]*phi[0]   +   P[1][0]*phi[1]   +
P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] + P[1][1]*phi[1] +
P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] + P[1][2]*phi[1] +
P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] + P[1][3]*phi[1] +
P[2][3]*phi[2] + P[3][3]*phi[3])))*(phi[0]*theta[0] - y[0] + phi[1]*theta[1] +
phi[2]*theta[2] + phi[3]*theta[3]);



P_New[0][0] = (P[0][0] - (P[0][0]*phi[0]*(P[0][0]*phi[0] + P[0][1]*phi[1] +
P[0][2]*phi[2]   +   P[0][3]*phi[3])   +   P[1][0]*phi[1]*(P[0][0]*phi[0]   +
P[0][1]*phi[1]      +      P[0][2]*phi[2]      +      P[0][3]*phi[3])      +
P[2][0]*phi[2]*(P[0][0]*phi[0]   +   P[0][1]*phi[1]   +   P[0][2]*phi[2]   +
P[0][3]*phi[3])   +   P[3][0]*phi[3]*(P[0][0]*phi[0]   +   P[0][1]*phi[1]   +
P[0][2]*phi[2]   +   P[0][3]*phi[3]))/(lambda[0]   +   phi[0]*(P[0][0]*phi[0]   +
```

```
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3]))/lambda[0];

P_New[0][1] = (P[0][1] - (P[0][1]*phi[0]*(P[0][0]*phi[0] + P[0][1]*phi[1] +
P[0][2]*phi[2]   +   P[0][3]*phi[3])   +   P[1][1]*phi[1]*(P[0][0]*phi[0]   +
P[0][1]*phi[1]      +      P[0][2]*phi[2]      +      P[0][3]*phi[3])      +
P[2][1]*phi[2]*(P[0][0]*phi[0]   +   P[0][1]*phi[1]   +   P[0][2]*phi[2]   +
P[0][3]*phi[3])   +   P[3][1]*phi[3]*(P[0][0]*phi[0]   +   P[0][1]*phi[1]   +
P[0][2]*phi[2]  +  P[0][3]*phi[3]))/(lambda[0]  +  phi[0]*(P[0][0]*phi[0]  +
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3]))/lambda[0];

P_New[0][2] = (P[0][2] - (P[0][2]*phi[0]*(P[0][0]*phi[0] + P[0][1]*phi[1] +
P[0][2]*phi[2]   +   P[0][3]*phi[3])   +   P[1][2]*phi[1]*(P[0][0]*phi[0]   +
P[0][1]*phi[1]      +      P[0][2]*phi[2]      +      P[0][3]*phi[3])      +
P[2][2]*phi[2]*(P[0][0]*phi[0]   +   P[0][1]*phi[1]   +   P[0][2]*phi[2]   +
P[0][3]*phi[3])   +   P[3][2]*phi[3]*(P[0][0]*phi[0]   +   P[0][1]*phi[1]   +
P[0][2]*phi[2]  +  P[0][3]*phi[3]))/(lambda[0]  +  phi[0]*(P[0][0]*phi[0]  +
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3]))/lambda[0];

P_New[0][3] = (P[0][3] - (P[0][3]*phi[0]*(P[0][0]*phi[0] + P[0][1]*phi[1] +
P[0][2]*phi[2]   +   P[0][3]*phi[3])   +   P[1][3]*phi[1]*(P[0][0]*phi[0]   +
P[0][1]*phi[1]      +      P[0][2]*phi[2]      +      P[0][3]*phi[3])      +
P[2][3]*phi[2]*(P[0][0]*phi[0]   +   P[0][1]*phi[1]   +   P[0][2]*phi[2]   +
P[0][3]*phi[3])   +   P[3][3]*phi[3]*(P[0][0]*phi[0]   +   P[0][1]*phi[1]   +
P[0][2]*phi[2]  +  P[0][3]*phi[3]))/(lambda[0]  +  phi[0]*(P[0][0]*phi[0]  +
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3]))/lambda[0];

P_New[1][0] = (P[1][0] - (P[0][0]*phi[0]*(P[1][0]*phi[0] + P[1][1]*phi[1] +
P[1][2]*phi[2]   +   P[1][3]*phi[3])   +   P[1][0]*phi[1]*(P[1][0]*phi[0]   +
P[1][1]*phi[1]      +      P[1][2]*phi[2]      +      P[1][3]*phi[3])      +
P[2][0]*phi[2]*(P[1][0]*phi[0]   +   P[1][1]*phi[1]   +   P[1][2]*phi[2]   +
P[1][3]*phi[3])   +   P[3][0]*phi[3]*(P[1][0]*phi[0]   +   P[1][1]*phi[1]   +
P[1][2]*phi[2]  +  P[1][3]*phi[3]))/(lambda[0]  +  phi[0]*(P[0][0]*phi[0]  +
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3]))/lambda[0];

P_New[1][1] = (P[1][1] - (P[0][1]*phi[0]*(P[1][0]*phi[0] + P[1][1]*phi[1] +
P[1][2]*phi[2]   +   P[1][3]*phi[3])   +   P[1][1]*phi[1]*(P[1][0]*phi[0]   +
P[1][1]*phi[1]      +      P[1][2]*phi[2]      +      P[1][3]*phi[3])      +
P[2][1]*phi[2]*(P[1][0]*phi[0]   +   P[1][1]*phi[1]   +   P[1][2]*phi[2]   +
P[1][3]*phi[3])   +   P[3][1]*phi[3]*(P[1][0]*phi[0]   +   P[1][1]*phi[1]   +
P[1][2]*phi[2]  +  P[1][3]*phi[3]))/(lambda[0]  +  phi[0]*(P[0][0]*phi[0]  +
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3]))/lambda[0];

P_New[1][2] = (P[1][2] - (P[0][2]*phi[0]*(P[1][0]*phi[0] + P[1][1]*phi[1] +
P[1][2]*phi[2]   +   P[1][3]*phi[3])   +   P[1][2]*phi[1]*(P[1][0]*phi[0]   +
P[1][1]*phi[1]      +      P[1][2]*phi[2]      +      P[1][3]*phi[3])      +
```

```
P[2][2]*phi[2]*(P[1][0]*phi[0]   +   P[1][1]*phi[1]   +   P[1][2]*phi[2]   +
P[1][3]*phi[3])   +   P[3][2]*phi[3]*(P[1][0]*phi[0]   +   P[1][1]*phi[1]   +
P[1][2]*phi[2]   +   P[1][3]*phi[3]))/(lambda[0]   +   phi[0]*(P[0][0]*phi[0]   +
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])))/lambda[0];

P_New[1][3] = (P[1][3] - (P[0][3]*phi[0]*(P[1][0]*phi[0] + P[1][1]*phi[1] +
P[1][2]*phi[2]   +   P[1][3]*phi[3])   +   P[1][3]*phi[1]*(P[1][0]*phi[0]   +
P[1][1]*phi[1]      +      P[1][2]*phi[2]      +      P[1][3]*phi[3])      +
P[2][3]*phi[2]*(P[1][0]*phi[0]   +   P[1][1]*phi[1]   +   P[1][2]*phi[2]   +
P[1][3]*phi[3])   +   P[3][3]*phi[3]*(P[1][0]*phi[0]   +   P[1][1]*phi[1]   +
P[1][2]*phi[2]   +   P[1][3]*phi[3]))/(lambda[0]   +   phi[0]*(P[0][0]*phi[0]   +
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])))/lambda[0];

P_New[2][0] = (P[2][0] - (P[0][0]*phi[0]*(P[2][0]*phi[0] + P[2][1]*phi[1] +
P[2][2]*phi[2]   +   P[2][3]*phi[3])   +   P[1][0]*phi[1]*(P[2][0]*phi[0]   +
P[2][1]*phi[1]      +      P[2][2]*phi[2]      +      P[2][3]*phi[3])      +
P[2][0]*phi[2]*(P[2][0]*phi[0]   +   P[2][1]*phi[1]   +   P[2][2]*phi[2]   +
P[2][3]*phi[3])   +   P[3][0]*phi[3]*(P[2][0]*phi[0]   +   P[2][1]*phi[1]   +
P[2][2]*phi[2]   +   P[2][3]*phi[3]))/(lambda[0]   +   phi[0]*(P[0][0]*phi[0]   +
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])))/lambda[0];

P_New[2][1] = (P[2][1] - (P[0][1]*phi[0]*(P[2][0]*phi[0] + P[2][1]*phi[1] +
P[2][2]*phi[2]   +   P[2][3]*phi[3])   +   P[1][1]*phi[1]*(P[2][0]*phi[0]   +
P[2][1]*phi[1]      +      P[2][2]*phi[2]      +      P[2][3]*phi[3])      +
P[2][1]*phi[2]*(P[2][0]*phi[0]   +   P[2][1]*phi[1]   +   P[2][2]*phi[2]   +
P[2][3]*phi[3])   +   P[3][1]*phi[3]*(P[2][0]*phi[0]   +   P[2][1]*phi[1]   +
P[2][2]*phi[2]   +   P[2][3]*phi[3]))/(lambda[0]   +   phi[0]*(P[0][0]*phi[0]   +
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])))/lambda[0];

P_New[2][2] = (P[2][2] - (P[0][2]*phi[0]*(P[2][0]*phi[0] + P[2][1]*phi[1] +
P[2][2]*phi[2]   +   P[2][3]*phi[3])   +   P[1][2]*phi[1]*(P[2][0]*phi[0]   +
P[2][1]*phi[1]      +      P[2][2]*phi[2]      +      P[2][3]*phi[3])      +
P[2][2]*phi[2]*(P[2][0]*phi[0]   +   P[2][1]*phi[1]   +   P[2][2]*phi[2]   +
P[2][3]*phi[3])   +   P[3][2]*phi[3]*(P[2][0]*phi[0]   +   P[2][1]*phi[1]   +
P[2][2]*phi[2]   +   P[2][3]*phi[3]))/(lambda[0]   +   phi[0]*(P[0][0]*phi[0]   +
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])))/lambda[0];

P_New[2][3] = (P[2][3] - (P[0][3]*phi[0]*(P[2][0]*phi[0] + P[2][1]*phi[1] +
P[2][2]*phi[2]   +   P[2][3]*phi[3])   +   P[1][3]*phi[1]*(P[2][0]*phi[0]   +
P[2][1]*phi[1]      +      P[2][2]*phi[2]      +      P[2][3]*phi[3])      +
P[2][3]*phi[2]*(P[2][0]*phi[0]   +   P[2][1]*phi[1]   +   P[2][2]*phi[2]   +
P[2][3]*phi[3])   +   P[3][3]*phi[3]*(P[2][0]*phi[0]   +   P[2][1]*phi[1]   +
P[2][2]*phi[2]   +   P[2][3]*phi[3]))/(lambda[0]   +   phi[0]*(P[0][0]*phi[0]   +
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])))/lambda[0];
```

```
P_New[3][0] = (P[3][0] - (P[0][0]*phi[0]*(P[3][0]*phi[0] + P[3][1]*phi[1] +
P[3][2]*phi[2]   +   P[3][3]*phi[3])   +   P[1][0]*phi[1]*(P[3][0]*phi[0]   +
P[3][1]*phi[1]      +      P[3][2]*phi[2]      +      P[3][3]*phi[3])      +
P[2][0]*phi[2]*(P[3][0]*phi[0]   +   P[3][1]*phi[1]   +   P[3][2]*phi[2]   +
P[3][3]*phi[3])   +   P[3][0]*phi[3]*(P[3][0]*phi[0]   +   P[3][1]*phi[1]   +
P[3][2]*phi[2]  +  P[3][3]*phi[3]))/(lambda[0]  +  phi[0]*(P[0][0]*phi[0]  +
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])))/lambda[0];

P_New[3][1] = (P[3][1] - (P[0][1]*phi[0]*(P[3][0]*phi[0] + P[3][1]*phi[1] +
P[3][2]*phi[2]   +   P[3][3]*phi[3])   +   P[1][1]*phi[1]*(P[3][0]*phi[0]   +
P[3][1]*phi[1]      +      P[3][2]*phi[2]      +      P[3][3]*phi[3])      +
P[2][1]*phi[2]*(P[3][0]*phi[0]   +   P[3][1]*phi[1]   +   P[3][2]*phi[2]   +
P[3][3]*phi[3])   +   P[3][1]*phi[3]*(P[3][0]*phi[0]   +   P[3][1]*phi[1]   +
P[3][2]*phi[2]  +  P[3][3]*phi[3]))/(lambda[0]  +  phi[0]*(P[0][0]*phi[0]  +
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])))/lambda[0];

P_New[3][2] = (P[3][2] - (P[0][2]*phi[0]*(P[3][0]*phi[0] + P[3][1]*phi[1] +
P[3][2]*phi[2]   +   P[3][3]*phi[3])   +   P[1][2]*phi[1]*(P[3][0]*phi[0]   +
P[3][1]*phi[1]      +      P[3][2]*phi[2]      +      P[3][3]*phi[3])      +
P[2][2]*phi[2]*(P[3][0]*phi[0]   +   P[3][1]*phi[1]   +   P[3][2]*phi[2]   +
P[3][3]*phi[3])   +   P[3][2]*phi[3]*(P[3][0]*phi[0]   +   P[3][1]*phi[1]   +
P[3][2]*phi[2]  +  P[3][3]*phi[3]))/(lambda[0]  +  phi[0]*(P[0][0]*phi[0]  +
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])))/lambda[0];

P_New[3][3] = (P[3][3] - (P[0][3]*phi[0]*(P[3][0]*phi[0] + P[3][1]*phi[1] +
P[3][2]*phi[2]   +   P[3][3]*phi[3])   +   P[1][3]*phi[1]*(P[3][0]*phi[0]   +
P[3][1]*phi[1]      +      P[3][2]*phi[2]      +      P[3][3]*phi[3])      +
P[2][3]*phi[2]*(P[3][0]*phi[0]   +   P[3][1]*phi[1]   +   P[3][2]*phi[2]   +
P[3][3]*phi[3])   +   P[3][3]*phi[3]*(P[3][0]*phi[0]   +   P[3][1]*phi[1]   +
P[3][2]*phi[2]  +  P[3][3]*phi[3]))/(lambda[0]  +  phi[0]*(P[0][0]*phi[0]  +
P[1][0]*phi[1] + P[2][0]*phi[2] + P[3][0]*phi[3]) + phi[1]*(P[0][1]*phi[0] +
P[1][1]*phi[1] + P[2][1]*phi[2] + P[3][1]*phi[3]) + phi[2]*(P[0][2]*phi[0] +
P[1][2]*phi[1] + P[2][2]*phi[2] + P[3][2]*phi[3]) + phi[3]*(P[0][3]*phi[0] +
P[1][3]*phi[1] + P[2][3]*phi[2] + P[3][3]*phi[3])))/lambda[0];


/**********************/
/* Guardando estados. */
/**********************/

xD[0]=theta_New[0];xD[1]=theta_New[1];xD[2]=theta_New[2];xD[3]=theta_New[3];

xD[4]=P_New[0][0];  xD[5]=P_New[0][1];  xD[6]=P_New[0][2];  xD[7]=P_New[0][3];
xD[8]=P_New[1][0];           xD[9]=P_New[1][1];           xD[10]=P_New[1][2];
xD[11]=P_New[1][3];
xD[12]=P_New[2][0];          xD[13]=P_New[2][1];          xD[14]=P_New[2][2];
xD[15]=P_New[2][3];
xD[16]=P_New[3][0];          xD[17]=P_New[3][1];          xD[18]=P_New[3][2];
xD[19]=P_New[3][3];
```